

UNIGRAPHICS

GRIP FUNDAMENTALS
STUDENT GUIDE
NOVEMBER 2003
MT13010 – Unigraphics NX 2

EDS Inc.

Proprietary & Restricted Rights Notices

Copyright

Proprietary right of Unigraphics Solutions Inc., its subcontractors, or its suppliers are included in this software, in the data, documentation, or firmware related thereto, and in information disclosed therein. Neither this software, regardless of the form in which it exists, nor such data, information, or firmware may be used or disclosed to others for any purpose except as specifically authorized in writing by Unigraphics Solutions Inc. Recipient by accepting this document or utilizing this software agrees that neither this document nor the information disclosed herein nor any part thereof shall be reproduced or transferred to other documents or used or disclosed to others for manufacturing or any other purpose except as specifically authorized in writing by Unigraphics Solutions Inc.

©2003 Electronic Data Systems Corporation. All rights reserved.

Restricted Rights Legend

The commercial computer software and related documentation are provided with restricted rights. Use, duplication or disclosure by the U.S. Government is subject to the protections and restrictions as set forth in the Unigraphics Solutions Inc. commercial license for the software and/or documentation as prescribed in DOD FAR 227–7202–3(a), or for Civilian Agencies, in FAR 27.404(b)(2)(i), and any successor or similar regulation, as applicable. Unigraphics Solutions Inc., 10824 Hope Street, Cypress, CA 90630.

Warranties and Liabilities

All warranties and limitations thereof given by Unigraphics Solutions Inc. are set forth in the license agreement under which the software and/or documentation were provided. Nothing contained within or implied by the language of this document shall be considered to be a modification of such warranties.

The information and the software that are the subject of this document are subject to change without notice and should not be considered commitments by Unigraphics Solutions Inc.. Unigraphics Solutions Inc. assumes no responsibility for any errors that may be contained within this document.

The software discussed within this document is furnished under separate license agreement and is subject to use only in accordance with the licensing terms and conditions contained therein.

Trademarks

EDS, the EDS logo, I–DEAS, UNIGRAPHICS SOLUTIONS®, **UNIGRAPHICS®**, **GRIP®**, **PARASOLID®**, **UG®**, **UG/...®**, **UG SOLUTIONS®**, **iMAN®** are trademarks or registered trademarks of Electronic Data Systems Corporation or its subsidiaries. All other logos or trademarks used herein are the property of their respective owners.

Grip Fundamentals Student Guide Publication History:

Version	16.0	January 2000
Version	17.0	December 2000
Version	18.0	February 2002
Unigraphics NX		December 2002
Unigraphics NX 2		November 2003

Table of Contents

Introduction	-1
Course Overview	-1
GRIP Applications	-2
Course Objectives	-3
GRIP Development Process	1-1
File Naming Conventions	1-2
Creating a GRIP Program	1-2
Program Development using GRADE	1-3
Edit	1-4
Compiling	1-5
Sections of a GRIP Compiler Listing	1-7
Errors	1-8
Linking	1-9
Running a Program	1-10
Activity: Sample GRIP program with errors	1-11
Activity: Edit a GRIP program	1-15
GRIP Language Components	2-1
GRIP Program Organization	2-2
GRIP Command Structure	2-3
GRIP Vocabulary	2-4
Major Words	2-4
Minor Words	2-4
Global Parameter Access Symbols (GPA's)	2-4
Entity Data Access Symbols (EDA's)	2-5
Conventions Used in this Manual	2-6
Declaring Variables	2-7
Numeric Variables	2-7
Entity Variables	2-12
String Variables	2-13
Initializing and Setting Variables	2-16
Assigning Values to String Variables	2-17
Assigning Values to Numeric Variables	2-18
Assigning Values to Entity Variables	2-19
Subrange Operators	2-20

Rules for Using Subrange Operators	2–21
Detect Undeclared Variables	2–23
Exercises	2–25
Two Dimensional Commands	3–1
Positional Modifiers	3–2
Two-Dimensions – PMOD2	3–2
Three-Dimensions – PMOD3	3–3
Point	3–4
Point: Coordinates	3–5
Point: Center of a Circle	3–6
Point: End Point	3–7
Point: Intersection Point	3–8
Point: EDA	3–10
Line	3–11
Line: Between Two Specified Points (Coordinates)	3–12
Line: Between Two Existing Points	3–13
Line: Parallel at a Distance	3–14
Line: Through a Point, At an Angle	3–16
Line: Point, Tangent to a Curve	3–18
Line: Through a Point, Parallel/Perpendicular to a Line ..	3–20
Line EDA's	3–22
Activity The L-shape	3–24
More Two Dimensional Commands	4–1
Circle	4–2
Circle: Center Coordinates, Radius	4–3
Circle: Center Point, Radius	4–5
Circle: Center Point, Tangent to a Line	4–7
Circle: Center Point, Point on the Arc	4–9
Circle: Through Three Points	4–11
Fillet	4–12
Fillet: Two Objects, Center Point	4–13
Fillet: Two Lines, Positional Modifiers	4–16
Circle EDA's	4–19
Entity Independent EDA's	4–21
Nesting	4–23
Activity: Sheet Metal Part	4–24
Controlling Program Execution	5–1
Introduction	5–2
Suggested Labeling Guidelines	5–3

JUMP: Unconditional Branching	5–6
JUMP: Conditional Branching	5–7
DO: Program Loop	5–9
IF: Logical IF	5–12
BLOCKIF: Block If	5–16
IF Boolean Operators	5–21
Error Handling	5–23
Basic Interactive Commands	6–1
Interactive Commands	6–2
Typical Interactive Menu Flow	6–5
Program Outline using Interactive Statements	6–8
POS: Indicate Screen Position Point	6–11
GPOS: Indicate Generic Point Position	6–13
PARAM: Enter Parameters	6–15
Activity: Interactive L–Shape	6–18
Advanced Interactive Commands	7–1
Interactive Commands	7–2
IDENT: Select Objects	7–3
TEXT: Enter Text	7–8
CHOOSE: Choose Single Option	7–11
MCHOOS: Choose Multiple Options	7–15
Activity: Interactive Sheet Metal Part	7–18
Solid Object Modeling Commands	8–1
Solid Feature Creation	8–2
Solid Operations	8–12
Solid Object EDAs	8–18
Solid Object GPAs	8–18
Activity: Extrude the L–Shape	8–19
Activity: Solid L–Shape	8–20
Database Cycling	9–1
File Access	9–2
Object Cycling	9–9
INEXTE: Initialize Data Model Cycling	9–11
NEXTE: Cycle to Next Object	9–12
INEXTN: Initialize Non–geometric Object Cycling	9–13
NEXTN: Cycle to Next Non–geometric Object	9–15
Non-Geometric Database Cycling Example	9–16
Reading and Writing to Text Files	9–17

READ: Read Text from a File	9–19
WRITE: Write Text to a File	9–21
PRINT: Print Data on Listing Device	9–22
Activity: Reading a Text File	9–23
Object Access	10–1
Categories and Layer Control	10–2
LAYER: Layer Control	10–3
CAT: Create Layer Category	10–5
CATE: Edit Category	10–7
CATD: Delete Category	10–8
CATV: Query Category	10–9
Object Control	10–10
MASK: Class Selection	10–12
TYPF: Object Type	10–15
OBTAIN: Object Information	10–17
Object Display Control	10–19
DRAW: Control Object Display	10–20
DELETE: Delete	10–23
Global Parameter Access (GPA) Symbols	10–24
Access Type (Read and Write Options) of GPA Functions	10–24
Data Types Assigned to GPAs	10–24
GPA Range	10–26
System Constants	10–26
Active Part Status	10–28
Decimal Places	10–28
Work Layer	10–29
Work View	10–29
Unit of Measurement	10–30
User ID	10–30
Activity: Entity Cycling	10–31
Transformations	11–1
Introduction	11–2
TRANSF: Transformation	11–4
MATRIX: Translate	11–6
MATRIX: Scale	11–7
MATRIX: Mirror	11–8
MATRIX: Rotate	11–9
MATRIX: Multiple Transformations (Concatenation)	11–10
Activity Transforming the L-Shape	11–12
Functions	12–1
Processing Arithmetic Expressions	12–2

Mathematical Functions	12–3
Arithmetic Functions	12–5
Trigonometric Functions	12–7
String Functions	12–9
Return Current Date	12–10
Set Full Date Flag	12–11
Return Current Time	12–12
Create Blank Characters	12–13
Number of Characters in a String	12–14
Convert Integer to Character String	12–15
Convert Real Number to Character String	12–16
Convert String to Real Number	12–17
ASCII Value of a Character	12–18
Return string with ASCII Value of N	12–19
Replace Characters in a String	12–20
Extract a portion of a Character String	12–22
Position of Characters in a String	12–23
Activity: Date Conversion	12–24
Activity: Case Conversion (Optional)	12–25
Subroutines	13–1
Subroutine Programming	13–2
Call Statement	13–3
Procedure Statement	13–4
Return Statement	13–6
Activity: Case Converter Subroutine	13–8
Drafting Functions	14–1
Drawing Overview	14–3
Create a Drawing	14–4
Change Drawing Size	14–5
Add View to Drawing	14–6
Verify a Drawing	14–7
Delete a Drawing	14–11
Current Drawing	14–12
Import an Existing Part	14–13
Import an Existing Part Grouped	14–16
Activity: Drawing Creation	14–19
Dimensions and Drafting Aids	15–1
Dimensions	15–2
Creating Drafting Objects on Drawings	15–5

Horizontal and Vertical Dimensions	15–8
Angular Dimensions	15–11
Radius Dimensions	15–13
Dimension Statements	15–15
Extension Line Display	15–18
Text and Arrow Location	15–19
Entity Site related to Text	15–20
Drafting Aids	15–22
Note	15–23
Label	15–25
Linear Centerline	15–27
Drafting EDAs: Drafting Object Origin	15–29
Drafting EDAs: Text	15–30
Activity: Dimensioning the L-shape	15–31
Attributes (Optional)	16–1
Object Names	16–2
&NAME	16–3
ENUM	16–5
&ENAME	16–6
DELNAM	16–8
ASATT	16–9
&ATTTL	16–12
&ATTVL	16–16
DLATT	16–19
&ATDISL	16–22
ASCII Conversion Table	A–1
GRIP Debugger	B–1
GRIP Debugger	B–2
Running a Program with the Debugger	B–3
Set Breakpoints	B–4
Clear Breakpoints	B–6
List Options	B–9
Final Project	C–1
Activity: Creating a Spur Gear	C–2
Statement Format Summary	D–1
GPA Symbol Format Summary	E–1
EDA Symbol Format Summary	F–1
Word/Symbol List	G–1
Index	IN–1

Introduction

Course Overview

The GRIP Fundamentals Course teaches the concepts of the Unigraphics Graphics Interactive Programming (GRIP) language.

How to Use this Book

This student guide is intended to be the classroom workbook. It is not intended to be a complete reference. For full reference detail consult the *UG/Open GRIP Reference Manual*.

This workbook is arranged as a series of lessons. Each lesson starts with general concepts and builds to detailed concepts. Each concept, whether it introduces a GRIP command or theory, includes a description of the concept, the command for the particular application, and an illustration of the concept. Examples and sample programs in this workbook demonstrate these concepts.

GRIP

GRIP is a software package developed by Unigraphics Solutions. GRIP is used to create FORTRAN–like programs to operate the Unigraphics system. Many operations that can be performed interactively can be performed by using a GRIP program. Commands are available to create geometric and drafting entities, control preferences, perform file management functions and modify existing geometry. GRIP also provides interactive commands. These commands display messages in a motif dialog, allowing the user to interact with a GRIP program while it is running. The messages displayed by these commands can be programmed to fit the user’s specific needs. There are interactive commands to control entity selection, menu option selection, data entry, text entry and the Generic Point Subfunction.

GRIP Applications

GRIP/Unigraphics

- Special Geometric Functions
- Calculation and Analysis
- Part Standardization
- Family–of–Parts Programming
- File Management
- Data Access

GRIP N/C Machining

- Basic Machining
- Planar Milling
- Surface Milling
- Lathe Machining

GRIP/Finite Element Modeling

- Node Creation and Editing
- Element Creation and Editing
- Mesh Node/Element Creation
- Mesh Definition by Surface
- CNL Creation
- Node/Element Transformations

Intended Audience

This training manual is for students in the GRIP Fundamentals class.

Prerequisites

The student should be familiar with Unigraphics and its file naming conventions, and with fundamental programming techniques.

Course Objectives

- Create programs using a variety of GRIP commands and data types.
- Automate the construction of 2D wireframe and 3D solid geometry.
- Organize program execution using logical statements and branching commands in conjunction with appropriate error handling techniques.
- Develop user interfaces that request data.
- Examine and edit data stored in a Unigraphics part file.
- Copy and move geometry using transformations.
- Use string and mathematical functions.
- Create and use subroutines to manage program complexity.
- Effectively use the GRIP Debugger to reduce development time.
- Create and modify drawings.
- Create notes, labels, and dimensions.

(This Page Intentionally Left Blank)

GRIP Development Process

Lesson 1



Objectives

- Demonstrate an understanding of the rules for creating source files and file extensions used for GRIP programs.
- Run GRIP programs.
- Use an editor to create and edit GRIP source code.
- Recognize and fix compilation errors; then successfully compile, link, and run a GRIP program.

File Naming Conventions

A prerequisite for this class is that you are familiar with Unigraphics and Unigraphics file naming conventions. Table 1–1 lists the file types and file extensions associated with GRIP files.

Table 1–1 GRIP File, File Name Extension, and File Type

GRIP File	File Name* and Extension	File Type
Source Code	name.grs	Text
Object Code	name.gri	UGII
Executable Code	name.grx	GRXI

* In Table 1–1 above, the word “name” refers to any name assigned to a program.

Creating a GRIP Program

Follow these steps to create and run a GRIP program:

1. Develop and input the source file for the GRIP program using an editor.
2. Once the source file has been created, compile the GRIP program. The GRIP compiler converts the GRIP source file (.grs extension) into a GRIP object file (.gri extension).
3. After compilation has been successfully completed, use the Link option to link the GRIP object file. This combines the main program with all the subprograms (if any) and creates a GRIP execution file (.grx extension).
4. The GRIP execution (.grx) file can now be run interactively after entering Unigraphics or through GRIP batch processing.

Program Development using GRADE

The development of a GRIP program or subprogram begins when the source file is created. The source file consists of statements, labels, and comments arranged in a logical manner to perform various tasks.

This file is created by using an operating system editor, typically after entering GRADE, the GRIP Advanced Development Environment. GRADE facilitates editing, compiling, and linking to create a GRIP executable file. The executable file can then be run after entering Unigraphics or by using GRIP batch.

GRIP Advanced Development Environment (GRADE) is a separate, executable program that allows a variety of GRIP program development functions to be performed from the operating system. These functions include those on the menu below:

GRIP Advanced Development Environment

- | | |
|---------------------|--------------------------|
| 1) Edit | 6) send Output to [CRT] |
| 2) Compile | 7) comPile listing [ALL] |
| 3) Link | 8) change ediTor [vi] |
| 4) change Directory | 9) grade Batch |
| 5) liSt directory | 0) turn Menu on/off |
- q) QUIT

The GRADE commands can be invoked in one of two ways, either by entering the number or upper case character for the desired option. For instance, the **Link** option can be used by entering either a “3” or a “L”.

Edit

GRIP Advanced Development Environment

- | | |
|----------------------------|---------------------------------|
| 1) Edit | 6) send Output to [CRT] |
| 2) Compile | 7) comPile listing [ALL] |
| 3) Link | 8) change ediTor [vi] |
| 4) change Directory | 9) grade Batch |
| 5) liSt directory | 0) turn Menu on/off |
- q) **QUIT**

To Edit, enter a “1” or an “E”. This will invoke the specified operating system editor after the file name is specified.

Enter the file name using the following guidelines:

- The file name may contain up to 26 characters before the “.grs” extension.
- Any alpha or numeric character may be used.
- The characters “-”, and “_” are permitted. All other special characters should not be used.

When editing the source file through GRADE, the “.grs” file extension will automatically be added to the entered file name. Alternatively, if the source file is created directly from an operating system prompt, be sure to specify the “.grs” extension in the file name, so that it will be properly recognized as a source code file.

Compiling

Once editing is complete and the GRIP source file is created, it can then be compiled.

Several things happen during the compilation process. First the GRIP compiler converts the GRIP source file (program or subprogram) into an object file (.gri extension). Each line of the GRIP source file is analyzed and a listing is generated which consists of statements, labels, variables and any errors recognized by the compiler. If the GRIP source file compiles without errors, the object file is automatically created and filed in the current directory. If errors exist, the object file is not created. Finally, the compiler lists the number of subprograms that were either compiled without error or failed compilation. If the program(s) failed compilation, the name(s) of the program(s) that failed will be listed.

GRIP Advanced Development Environment

- | | |
|---------------------|--------------------------|
| 1) Edit | 6) send Output to [CRT] |
| 2) Compile | 7) comPile listing [ALL] |
| 3) Link | 8) change ediTor [vi] |
| 4) change Directory | 9) grade Batch |
| 5) liSt directory | 0) turn Menu on/off |
| q) QUIT | |

To compile, select option “2” or “C” on the GRADE menu. This starts the compile operation.

If a file name has been previously entered, this file name will be assumed as the default file name. A file name may also be entered, if it is not desired to compile the current file.

Multiple files can be compiled by specifying a file name using the “*” character as a template. For example, if all source files in a directory are to be compiled, enter a file specification of “*.grs”.

After compiling the source file, a listing may be displayed of the GRIP source statements, variable storage locations, labels, errors, etc. or a listing of the errors only. To toggle between a complete listing and an errors only listing, use option “7” or “P” as indicated on the following GRADE menu. The default is to list all of the compiler output.

GRIP Advanced Development Environment

- | | |
|---------------------|--------------------------|
| 1) Edit | 6) send Output to [CRT] |
| 2) Compile | 7) comPile listing [ALL] |
| 3) Link | 8) change ediTor [vi] |
| 4) change Directory | 9) grade Batch |
| 5) liSt directory | 0) turn Menu on/off |

q) QUIT

On the following page is a typical listing of a successful compilation. An explanation of the information in the listing is provided below:

- Notice that the heading shows the program name and compiler version.
- The program listing appears with each line of the source code numbered (Option 7 was set to ALL).
- Following the program listing, the name, type, address, and dimensions of the variables will appear.
- Next the description of the processing of the information appears.
- The program ends with a “HALT” statement.
- The listing of the number of errors, the program size and the data size appears for each program compiled.
- Finally, the number of programs compiled without errors and number of programs that failed compilation along with a list of those programs which failed compilation is provided.

At the end of this Lesson, you will complete an exercise to practice the GRIP Development Process after editing a program which contains compilation errors.

Sections of a GRIP Compiler Listing

	=====																				
1. Heading:	UNIGRAPHICS GRIP COMPILER, REV 06																				
2. Body of Program:	PROGRAM : gf_sheet																				
3. Comments:	<pre> 1 \$ 2 \$\$ 3 \$\$ Description: This program creates 4 \$\$ 2-dimensional geometry 5 \$\$ to represent a part edge 6 \$\$ and a bolt 7 \$\$ 8 \$ 9 10 ENTITY/ edge(4), bolt(2), e_group, b_group 11 12 \$\$ 13 \$\$ Create the edge. 14 \$\$ 15 16 edge(1) = LINE/ -1, -.5, 1, -.5 17 edge(2) = LINE/ 1, -.5, 1, .5 18 edge(3) = LINE/ 1, .5, -1, .5 19 edge(4) = LINE/ -1, .5, -1, -.5 20 e_group = GROUP/ edge(1..4) 21 22 \$\$ 23 \$\$ Create the bolt. 24 \$\$ 25 26 bolt(1) = CIRCLE/ 0, 0, .5 27 bolt(2) = CIRCLE/ 0, 0, .25 28 b_group = GROUP/ bolt(1..2) 29 30 HALT </pre>																				
4. Declarations:																					
5. Halt Statement:																					
6. Compile information:	<table border="0"> <thead> <tr> <th>NAME</th> <th>TYPE</th> <th>ADDR</th> <th>DIM</th> </tr> </thead> <tbody> <tr> <td>EDGE</td> <td>ENT ARRAY</td> <td>0</td> <td>4</td> </tr> <tr> <td>BOLT</td> <td>ENT ARRAY</td> <td>4</td> <td>2</td> </tr> <tr> <td>E_GROU</td> <td>ENTITY</td> <td>6</td> <td></td> </tr> <tr> <td>B_GROU</td> <td>ENTITY</td> <td>7</td> <td></td> </tr> </tbody> </table>	NAME	TYPE	ADDR	DIM	EDGE	ENT ARRAY	0	4	BOLT	ENT ARRAY	4	2	E_GROU	ENTITY	6		B_GROU	ENTITY	7	
NAME	TYPE	ADDR	DIM																		
EDGE	ENT ARRAY	0	4																		
BOLT	ENT ARRAY	4	2																		
E_GROU	ENTITY	6																			
B_GROU	ENTITY	7																			
7. Errors, Program Size and Data Size:	<pre> 0 ERRORS, PROGRAM = 162, DATA = 8 UNIVERSAL LINK FILE ===== *** 1 GRIP PROGRAM COMPILED WITHOUT ERROR ===== </pre>																				



Errors

Program error messages may appear on the screen after attempting to Compile, Link or Run. Types of GRIP error messages are listed below. You can find a complete list of error messages listed in the *UG/Open GRIP Reference Manual*.

If your GRIP source file compiles without errors, the object file is automatically created and you return to the Grade menu. If errors exist, the object file is not created and a message indicating the number of files that failed compilation is displayed. You must then Edit the program(s) and correct the errors.

Error Range	Error Type
1–99	Non–fatal compilation errors. Compilation continues but object code cannot be filed.
100–199	Fatal compilation errors. Compilation is halted.
500–599*	Compiler system errors.
600–699*	Interpreter system errors.
1545000– 1545999	Non–fatal execution errors. Program execution may be continued.
1549000– 1549999	Fatal execution errors. Program execution is halted.

NOTE *Errors numbered 500–699 indicate a malfunction in Unigraphics and should be reported to Unigraphics Global Technical Access Center (GTAC). The toll free telephone number is: **1–800–955–0000**. You can also report problems via the World Wide Web. The site is **<http://support.ugsolutions.com>**. You must have a BBS account to log a problem report through the website.

Linking

As soon as compilation has been successfully completed, the executable program must be created through linking.



GRIP Advanced Development Environment

- | | |
|---------------------|--------------------------|
| 1) Edit | 6) send Output to [CRT] |
| 2) Compile | 7) comPile listing [ALL] |
| 3) Link | 8) change ediTor [vi] |
| 4) change Directory | 9) grade Batch |
| 5) liSt directory | 0) turn Menu on/off |
- q) QUIT

Select GRADE option “3” or “L”.

The GRIP linker links the object file of a main GRIP program together with the object file of any GRIP subprograms into an executable program (.grx extension). During this process the system generates a listing which consists of the main program name and the names of all subprograms (if any) in the order in which they are referenced.

NOTE Linking is necessary even when all of the statements for a program reside in one source code file.

Enter the name of the main program that is to be linked. As the linking process proceeds, the system will produce a listing containing the name of the main program, the names of all the subprograms that the main program calls, and any linker errors. See the following example of this listing.

Example 1–1. Linker Listing Without Errors

UNIGRAPHICS 1-16-96
GRIP LINK LIST

PROGRAM NAME	PROG SECT	PROG DISP	DATA SECT	DATA DISP
FLOWCHART	0	0	0	245
INTERACTION	0	315	1	140
PROCESS	1	248	2	99
OFF-PAGE	2	248	3	404
DECISION	4	194	5	86
SPECIAL	5	239	6	131
GUIDELINE	6	255	6	442

LINKING COMPLETE

PROGRAM SIZE =0006 0491
INTERACTIVE EXECUTION FILE

***1 GRIP PROGRAM(S) LINKED WITHOUT ERROR

Running a Program

After the GRIP program links without errors, the executable module will be automatically filed. The program can be run in either in the interactive mode or in the batch mode to create the output of the program.

To run a program interactively, select the following sequence of options starting at the Unigraphics menu bar:

File —>Execute UG/Open —>Grip

A File Selection menu will be displayed with a list of all GRIP executable files. After the desired selection is made, the program will begin execution.

NOTE The execution of a GRIP program may require an active Unigraphics part. Therefore, the selected program may not execute properly without first activating a part.

Activity: Sample GRIP program with errors



Study the program below and determine where you would have to edit the program to remove errors.

- Hints:*
- Fix all typographical errors. Look at program statements similar to the one that has the errors for correct spelling.
 - Look for missing or extra delimiters. These include commas and parentheses. A single missing character can cause several errors.
 - Look for a missing continuation character, “\$”. Without this character, the lines after it will error and so will the statement that would have contained the “\$”.

```

=====
UNIGRAPHICS GRIP COMPILER, REV 06
PROGRAM : sample_bad
      1  $$
      2  $$  DECLARE THE ENTITY NAMES TO BE USED IN
      3  $$  THE PROGRAM
      4  $$
      5      ENTITY/E(20),TMP(5)
      6      NUMBER/X0,Y0,Z0,WD,HT,TH,R1,R2,R3
      7  $$
      8  $$  DEFINE X Y Z VALUES TO LOCATE THE ORIGIN OF THE HANDLE
      9  $$  DEFINE THE X Y Z VALUES TO BE USED TO CONSTRUCT THE HANDLE
     10  $$
     11      DATA/X0,1,Y0,1.5,Z0,0
     12      DAAT/WD,5,HT,2.5,R1,.25,R2,.5,R3,.25,TH,.5
           *
Error no. 15 in line no. 12: Expression on left of equal sign.
     13  $$
     14  $$  DISPLAY/VERIFY DIMENSIONS:
     15  $$  (ALSO CHECK FOR ERRORS)
     16  $$
     17  BCK010:
     18      PARAM/'HANDLE CONSTRUCTION*DIMENSIONS', $
     19      'XC',X0,'YC',Y0,'ZC',Z0,
     20      'HEIGHT',HT,'WIDTH',WD,$
           *
Error no. 1 in line no. 20: Syntax error.
     21      'THICKNESS',TH,RESP
           *
Error no. 1 in line no. 21: Syntax error.
     22      JUMP/CANCEL:,CANCEL:,RESP
     23  $$
     24  $$  ERROR CHECKING:
     25  $$  (SET LIMITS FOR CONSTRUCTING A GOOD HANDLE)

```



```

26 $$
27     IFTHEN/TH<R1*2 OR TH>R2*2
28         MESSG/'INVALID THICKNESS', $
29         '(MUST BE'+FSTR(R1*2)+'<=T<='+FSTR(R2*2)+' )'
30         JUMP/BCK010:
31     ELESIF/HT<1.5
32         *
Error no. 15 in line no. 31: Expression on left of equal sign.
32         MESSG/'INVALID HEIGHT', '(MUST BE >=1.5)'
33         JUMP/BCK010:
34     ELSEIF/WD<3.5
35         MESSG/'INVALID HEIGHT', '(MUST BE >=3.5)'
36         JUMP/BCK010:
37     ENDIF
38 $$
39 $$ DEFINE THE GEOMETRY:
40 $$ (START AT LOWER LEFT ARC AND WORK AROUND
41 $$ COUNTER-CLOCKWISE).
42 $$
43     E(1)=CIRCLE/X0,Y0,Z0,R2,START,100,END,80
44     E(3)=LINE/(X0+TH/2),Y0,Z0,(X0+TH/2),Y0+HT,Z0
45     E(5)=LIN/X0,(Y0+HT),Z0,(X0+WD),(Y0+HT),
46         *
Error no. 1 in line no. 45: Syntax error.
46     E(7)=LIN/(X0+WD-TH/2),Y0,Z0,(X0+WD-TH/2),Y0+HT,Z0
47         *
Error no. 1 in line no. 46: Syntax error.
47     E(9)=CIRCL/(X0+WD),Y0,Z0,R2,START,100,END,
48         *
Error no. 1 in line no. 47: Syntax error.
48     E(11)=LINE/PARLEL,E(7),XLARGE,TH
49     E(13)=LINE/PARLEL,E(5),YLARGE,TH
50     E(15)=LINE/PARLEL,E(3),XSMALL,TH
51 $$
52 $$ CONSTRUCT TEMPORARY POINTS FOR FILLETING:
53 $$
54     TMP(1)=PNT/(X0+WD/2),(Y0+R2),Z0    $$ FOR INSIDE FILLETS
55         *
Error no. 1 in line no. 54: Syntax error.
55     TMP(2)=POINT/(X0+WD+TH),(Y0+R2),Z0    $$ FOR RIGHT OUTSIDE
56     TM(3)=POINT/(X0-TH),(Y0+R2),Z0    $$ FOR LEFT OUTSIDE
57         *
Error no. 1 in line no. 56: Syntax error.
57 $$
58 $$ NOW CONSTRUCT FILLETS:
59 $$ (CLOCKWISE AROUND OUTSIDE OUTLINE, TOO)
60 $$
61     E(2)=FILLET/E(3),E(1),CENTER,TMP(1),RADIUS,R3
62     E(4)=FILLET/YSMALL,E(5),XLARGE,E(3),RADIUS,R3,
63     E(6)=FILLET/XSMALL,E(7),YSMALL,E(5),RADIUS,R3
64     E(8)=FILLET/E(9),E(7),CENTER,TMP(1),RADIUS,R3
65     E(10)=FILLET/E(11),E(9),CENTER,TMP(2),RADIUS,R3
66     E(12)=FILL/XSMALL,E(11),YSMALL,E(13),RADIUS,R3+TH
67         *
Error no. 1 in line no. 66: Syntax error.

```



```

67      E(14)=FILL/YSMLL,E(13),XLARGE,E(15),RADIUS,R3+TH
          *
Error no. 1 in line no. 67: Syntax error.
68      E(16)=FILL/E(1),E(15),CENTER,TMP(3),RADIUS,R3
          *
Error no. 1 in line no. 68: Syntax error.
69      DELETE/TMP(1..3) $$ CLEAN TEMP POINTS
70      $$
71      $$ PUT HOLES IN HANDLE, AND GROUP IT ALL:
72      $$
73      E(17)=CIRCLE/X0,Y0,Z0,R1
74      E18)=CIRCLE/(X0+WD),Y0,Z0,R1
          *
Error no. 11 in line no. 74: Unbalanced parentheses in an arithmetic
expression.
75      GROUP/E(1..18)
76      $$
77      CANCEL:
78      HALT
Error no. 70 PARAM at argument no. 8 in line no. 18: Invalid field.
Error no. 71 (FILLET) in line no. 62: Too many fields.
Error no. 58 in line no. 68: Illegal use of division.
NAME      TYPE          ADDR          DIM
E          ENT ARRAY    0           20
TMP       ENT ARRAY    20          5
X0        NUMBER        25
Y0        NUMBER        27
Z0        NUMBER        29
WD        NUMBER        31
HT        NUMBER        33
TH        NUMBER        35
R1        NUMBER        37
R2        NUMBER        39
R3        NUMBER        41
DAAT     NUMBER        43
BCK010   LABEL          6
CANCEL   LABEL        703
RESP     NUMBER        73
ELESIF   NUMBER        93
LIN      NUMBER        119
CIRCL    NUMBER        121
PNT      NUMBER        123
TM       NUMBER        125
FILL     NUMBER        127
XSMLL    NUMBER        129
YSMLL    NUMBER        131
E18      NUMBER        133
16 ERRORS, PROGRAM = 708, DATA = 135
=====
*** 0 GRIP PROGRAMS COMPILED WITHOUT ERROR
*** 1 GRIP PROGRAM FAILED COMPILATION, IT IS:
sample_bad.grs
=====

```





(This Page Intentionally Left Blank)

Activity: Edit a GRIP program



This exercise is designed to provide you practice in editing a GRIP program. Edit the program on the following page to match the program that faces it.

Follow these instructions:

- Make a copy of the file using your initials.
(UNIX command: `cp parts/gf-intro.grs`
`./***-gf-intro.grs`)
Your instructor will discuss this step with the class.
- Enter GRADE and edit the file you copied.
- Use editing commands as needed to modify the program to be like the one on the facing page.
- After all edits are complete, save the file.
- Compile the program. NOTE: You may need to re-enter the file using the editor to make corrections if there are compilation errors.
- Link the program.
- Enter Unigraphics, **create a new part file**, and execute the program.

xxx-intro.grs (original)

The program below is the unedited version for the gf-intro activity.

```

$$ PROGRAM NAME:  gf-intro
$$ AUTHOR:       STAFF
$$ CREATION DATE: MAY 1999
$$ RELEASE:     UG NX
$$ PART ACTIVE:  YES
$$ SUBROUTINES: NONE
$$ ABSTRACT:    INTRODUCTION TO GRIP
$$
$$ *****
$$ DECLARATION AND INITIALIZATION
$$ *****
$$
$$
$$ Interactive prompt to ask user to locate a note
$$
BCK010:
  GPOS/'Specify a Position', $
      x, y, z, resp
  JUMP/BCK010:,ZZZ,,,,resp
$$
$$ Interactive prompt to ask for student name
$$
BCK020:
  TEXT/xxxxxxxxxxxxxxxx,s_name,resp,DEFLT
  JUMP/BCK010:,ZZZ,,,,resp
$$
$$ Set character size to .25
$$
  &CSIZE=.25
$$
$$ Set module parameters for entity site to be mid-center
$$
  &ENSITE=&MIDC
$$
$$ Create a note and place it in the center of the screen
$$
  welc_note = NOTE/x, y, $
              '<C2>' + s_name + $
              '<C1> Welcome to the <C3><F3>GRIP 1<C><F> CLASS'
$$
$$ AUTO MAX/MIN THE VIEW
$$
  VIEWE/AUTO
ENTITY/ xxxxxxxx  $$ move this line to after DECLARATION and INITIALIZATION
STRING/ s_name(30)  $$ move this line to after DECLARATION and INITIALIZATION
NUMBER/ x, y, z, resp  $$ move this line to after DECLARATION and INITIALIZATION
ZZZ
  HALT

```

xxx-intro.grs (solution)

The program below is the solution for the gf-intro activity

```

$$ PROGRAM NAME:  gf-intro
$$ AUTHOR:        STAFF
$$ CREATION DATE: MAY 1999
$$ RELEASE:      UG NX
$$ PART ACTIVE:  YES
$$ SUBROUTINES:  NONE
$$ ABSTRACT:     INTRODUCTION TO GRIP
$$
$$ *****
$$ DECLARATION AND INITIALIZATION
$$ *****
$$
ENTITY/ welc_note      $$ move this line to after DECLARATION and INITIALIZATION
STRING/ s_name(30)    $$ move this line to after DECLARATION and INITIALIZATION
NUMBER/ x, y, z, resp $$ move this line to after DECLARATION and INITIALIZATION
$$
$$      Interactive prompt to ask user to locate a note
$$
BCK010:
      GPOS/'Specify a Position', $
            x, y, z, resp
      JUMP/BCK010:,CANCEL:,,,,resp
$$
$$      Interactive prompt to ask for student name
$$
BCK020:
      TEXT/'Enter Your Name Please',s_name,resp,DEFLT
      JUMP/BCK010:,CANCEL:,,,,resp
$$
$$      Set character size to .25
$$
      &CSIZE=.25
$$
$$      Set module parameters for entity site to be mid-center
$$
      &ENSITE=&MIDC
$$
$$      Create a note and place it in the center of the screen
$$
      welc_note = NOTE/x, y, $
                  '<C2>' + s_name + $
                  '<C1> Welcome to the <C3><F3>GRIP 1<C><F> CLASS'
$$
$$ AUTO MAX/MIN THE VIEW
$$
      VIEWE/AUTO
CANCEL:
      HALT

```





(This Page Intentionally Left Blank)

GRIP Language Components

Lesson 2

Objectives

- Demonstrate knowledge of the style used to explain the GRIP language in the student guide.
- Demonstrate knowledge of the components of the GRIP language.
- Demonstrate knowledge of how a typical GRIP program is structured.
- Use initialization statements available in a GRIP program.



GRIP Program Organization

A GRIP source program is composed of a series of statements in the GRIP language. Generally, a GRIP program can be organized into five major areas listed below. Each area has related GRIP language commands.

GRIP Program Outline

Declaration Statements

ENTITY/
NUMBER/
STRING/

Initialization Statements

DATA/
Assignments
 Numeric (A=3)
 String (usr_name = 'Pat Smith')

Interactive Statements

PARAM/
TEXT/
GPOS/
etc.

Processing Statements

POINT/
LINE/
CIRCLE/
DELETE/
etc.

Termination

HALT

This lesson will provide a general focus on all five areas of a GRIP program, with detailed discussion of the first two areas, declaration and initialization statements.

GRIP Command Structure

Each line of a GRIP statement may contain no more than 80 characters. A single command may be split between several lines, thus creating a multiple line command. This is done by using the continuation character (\$) at the end of each line of the multiple line command, remembering that each line can contain no more than 80 characters.

CAUTION When the command is separated close to a comma (,) the comma must appear at the end of the line to be continued, before the continuation character (\$). This is illustrated in the example below:

Example

```
FILLET/ XSMALL, line1, YLARGE, line2, RADIUS, $  
      .5, NOTRIM
```

You may use blank characters (spaces) to separate the major and Minor words and variables in a command line.



GRIP Vocabulary

There are four types of GRIP reserved vocabulary words:

Example

```
&ENTCLR = &YELLOW  
ln1 = LINE/ 0, 0, 0, 10, 0, 0  
ln2 = LINE/ PARLEL, ln1, YSMALL, 12  
&COLOR(ln2) = &GREEN
```

A complete categorized list of the GRIP vocabulary words is provided in the Appendix. These vocabulary words should be treated as reserved words.

Major Words

GRIP command words, usually followed by the slash (/) character. In Example 2–2, LINE is a major word.

Some Major words are:

CALL, DELETE, CIRCLE, LINE, PARAM, HALT

Minor Words

Command modifiers used in conjunction with major words which describe various options available for a particular command. Minor words always follow the slash (/) character in the command line. In Example 2–2, PARLEL and YSMALL are minor words.

Some Minor words are:

ANGLE, ARROW, PARLEL, START, END, INTOF

Global Parameter Access Symbols (GPA's)

Enable various preferences which control the Unigraphics environment to be set. GPA's are always preceded by the ampersand (&) character. In Example 2–2, &ENTCLR is a GPA.

Some GPA's are:

&UNIT, &ACTPRT, &FONT

Entity Data Access Symbols (EDA's)

Enable information about entities to be extracted or edited. Like GPA's, EDA's are always preceded by the ampersand (&) character. In Example 2–2, &COLOR, &YELLOW, and &GREEN are EDA's.

Some EDA's are:

&LAYER, &COLOR, &POINT, &FONT



Conventions Used in this Manual

This manual describes GRIP statements using a shorthand convention that makes the statements easier to understand.

- GRIP vocabulary words are written in upper case letters.
- Numerical values and variables or expressions are written in lower case letters.

Example

Actual Statement	In This Book
SQRTF(B)	SQRTF(num)
STORE/3,A-11.3	STORE/reg,num

- Some of the GRIP statements have optional information. These optional fields are indicated by enclosing the field in square brackets.

Example

```
POINT/x,y[ ,z]
```

Example

```
FILLET/PMOD3,line1,PMOD3,line2,RADIUS,$  
r[ ,NOTRIM]
```

- Some statements have a choice of several possibilities in a specific field. These are indicated by placing the options in curly braces with vertical slashes (|) between. This convention means that you must select one of the options indicated between the vertical slashes from within the braces.

Example

```
FONT/{SOLID|DASH|PHANTM|CENTER}
```

The example shown above is how the manual would describe a GRIP statement but not the way you would write a statement in a program.

The GRIP language is not case sensitive so code can be written in either upper case or lower case. For clarity, it is recommended that you use the same convention used in this manual, with GRIP vocabulary words in upper case and user-defined variables in lower case.

Declaring Variables

Declaration statements are used to assign variables. The declaration statements reserve space in memory to store data in a GRIP program.

NOTE Declaration statements must precede the first line of executable code.

GRIP has the following three types of variables:

- ENTITY/ is used for geometric elements and other data created by Unigraphics.
- STRING/ is used for storing character data.
- NUMBER/ is used for storing both integer and real numerical data.

The rules for naming variables are listed below:

- The name cannot exceed 32 characters.
- The first six characters must be unique, i.e., cannot be the same as the first six characters of another variable name.
- The first character must be alphabetic, with the remaining characters alphanumeric.
- Characters may not be punctuation marks.
- The name should not be one of the reserved words.

Numeric Variables

The NUMBER word declares variables and arrays with up to three dimensions. GRIP does not distinguish between integer and real data. All numbers are stored in GRIP as double precision. Numbers *can* be used as if they were integers.

All subscripted arrays must be declared. Any variable used in a GRIP program but not declared as ENTITY, STRING or NUMBER is assumed to be a NUMBER. A GRIP switch can be used to override this default assumption.

There are two types of numerical variables, simple and subscripted.

- Simple Variable

This variable does not have to be declared in the NUMBER/ statement but it is a good practice to do so.

Example

In the following example, S3=15. Also, S1, S2, and S3 are three distinct variables.

```
S1=5  
S2=3  
S3=S1*S2
```

- Subscripted Variable

This is a numerical array of elements in one, two, or three dimensions. Arrays let you store several items of related information in a convenient fashion. This variable must be declared in the NUMBER statement.

In a declaration statement, a constant subscript within parenthesis indicates the array dimension. When the array is referenced throughout the code, a subscript indicates which portion of the array is being used in the GRIP statement. Throughout the code, the subscript may be a numerical value, expression, or subrange enclosed in parenthesis.

Example

In the following example, T is a one dimensional array consisting of three variable elements: 5, 3, and 15.

```
NUMBER/T(3)  
T(1)=5  
T(2)=3  
T(3)=T(1)*T(2)
```



Example One Dimension Variable Array

Format: NUMBER/Variable Name (Column)

```

NUMBER/NMS ( 10 )
NMS ( 1 ) = 2 . 1
NMS ( 2 ) = 5 . 6
NMS ( 3 ) = 7 . 8
NMS ( 4 ) = 9 . 2
.
.
.
NMS ( 10 ) = 2 . 9

```

	1	2	3	4	5	6	7	8	9	10
NMS	2.1	5.6	7.8	9.2	0	0	0	0	0	2.9

Figure 2–1 Graphic Illustration of a One Dimensional Variable Array

Example Two Dimensional Variable Array

Format: NUMBER/Variable Name (Column, Row)

```

NUMBER/NMS ( 5 , 2 )
NMS ( 1 , 1 ) = 2 . 1
NMS ( 2 , 1 ) = 7 . 8
NMS ( 1 , 2 ) = 5 . 6
NMS ( 2 , 2 ) = 9 . 2
.
.
.
NMS ( 5 , 2 ) = 2 . 9

```

NMS	1	2	3	4	5
1	2.1	7.8	0.0	0.0	0.0
2	5.6	9.2	0.0	0.0	2.9

Figure 2–2 Graphic Illustration of a Two Dimensional Variable Array

The terms *column* and *row* are conventions to describe a useful picture of a two dimensional array. You can use different terms and different pictures as long as you remember that GRIP assigns and stores values in arrays by varying the rightmost subscript most rapidly. It is important that you understand this principle when you use DATA/ to initialize an array.

Example Three Dimensional Variable Array

Format: NUMBER/Variable Name(Column, Row, Depth)

```

NUMBER/A,N(4,3,2)
DATA/N,1,2,3,4,5,6,7,8,9,10,11,12,13,$
      14,15,16,17,18,19,20,21,22,23,24
PRINT/N(3,2,2)
PRINT/N(1,3,1)
A = N(4,2,1) + N(1,3,2)
PRINT/A
HALT
    
```

Results of Print Commands

```

16.00
 5.00
27.00
    
```

Depth Storage Format

	Depth#1			
	COL#1	COL#2	COL#3	COL#4
ROW#1	1.00	7.00	13.00	19.00
ROW#2	3.00	9.00	15.00	21.00
ROW#3	5.00	11.00	17.00	23.00

	Depth#2			
	COL#1	COL#2	COL#3	COL#4
ROW#1	2.00	8.00	14.00	20.00
ROW#2	4.00	10.00	16.00	22.00
ROW#3	6.00	12.00	18.00	24.00

Figure 2-3 Data Storage Format

Table 2–1 Three Dimensional Array Data Assignment Table

$N(1,1,1)$	=	1.00
$N(1,1,2)$	=	2.00
$N(1,2,1)$	=	3.00
$N(1,2,2)$	=	4.00
$N(1,3,1)$	=	5.00
$N(1,3,2)$	=	6.00
$N(2,1,1)$	=	7.00
$N(2,1,2)$	=	8.00
$N(2,2,1)$	=	9.00
$N(2,2,2)$	=	10.00
$N(2,3,1)$	=	11.00
$N(2,3,2)$	=	12.00
$N(3,1,1)$	=	13.00
$N(3,1,2)$	=	14.00
$N(3,2,1)$	=	15.00
$N(3,2,2)$	=	16.00
$N(3,3,1)$	=	17.00
$N(3,3,2)$	=	18.00
$N(4,1,1)$	=	19.00
$N(4,1,2)$	=	20.00
$N(4,2,1)$	=	21.00
$N(4,2,2)$	=	22.00
$N(4,3,1)$	=	23.00
$N(4,3,2)$	=	24.00



Entity Variables

The ENTITY declaration establishes variables and arrays with up to three dimensions. An entity variable or array provides the ability to store and to reference objects that are created by the program, selected by the user, or accessed by cycling the data model.

Remember, you must make sure all declaration statements are before the first executable statement in the program. Entity variables are declared as follows:

ENTITY/name

ENTITY/name(dim1[,dim2][,dim3]),name...

where “name” is the variable name of the entity, and “dim1,dim2,dim3” define the array size.

If it is necessary for you to refer to a point (or any other entity) again in the program, you must give the point a name. This is called an “entity variable.”

Example

```
ENTITY/PT(3),LN4,P1,P2,LN3
P1 = POINT/ 3, 1, 6
P2 = POINT/ 2, 4, 6
LN4 = LINE/ P1, P2
PT(2) = POINT/ 3, 0, 0
PT(3) = POINT/ 4, 1, 2
LN3 = LINE/ PT(2), PT(3)
```

Notations such as P1, P2, and LN3, LN4 indicate single entities. There is no need to use parentheses when referencing them in a program, since they have not been declared using a subscript.

Notations such as PT(3) refer to multiple entities in arrays. PT(3) declares an array called PT holding three spaces for entity values .

String Variables

The `STRING` major word declares string variables and arrays with up to two dimensions. A string is a sequence of characters used whenever text is required in GRIP, such as in notes, dimensions, messages, etc.

`STRING/NAME(n)`

`STRING/NAME(dim1[,dim2],n)`

where: `NAME` is a variable name of the string and `dim1, dim2` define the array size.

`n` is the number of characters in the string. `n` must be less than or equal to 132.

Example

Declaration of a string variable (`TXT`) and a 2 dimensional string array (`ST`).

```
STRING/TXT(11),ST(2,2,5)
TXT = 'MY DOG SPOT'
ST(2,2) = 'MAY'
ST(1,2) = 'JUNE'
ST(1,1) = 'APRIL'
```

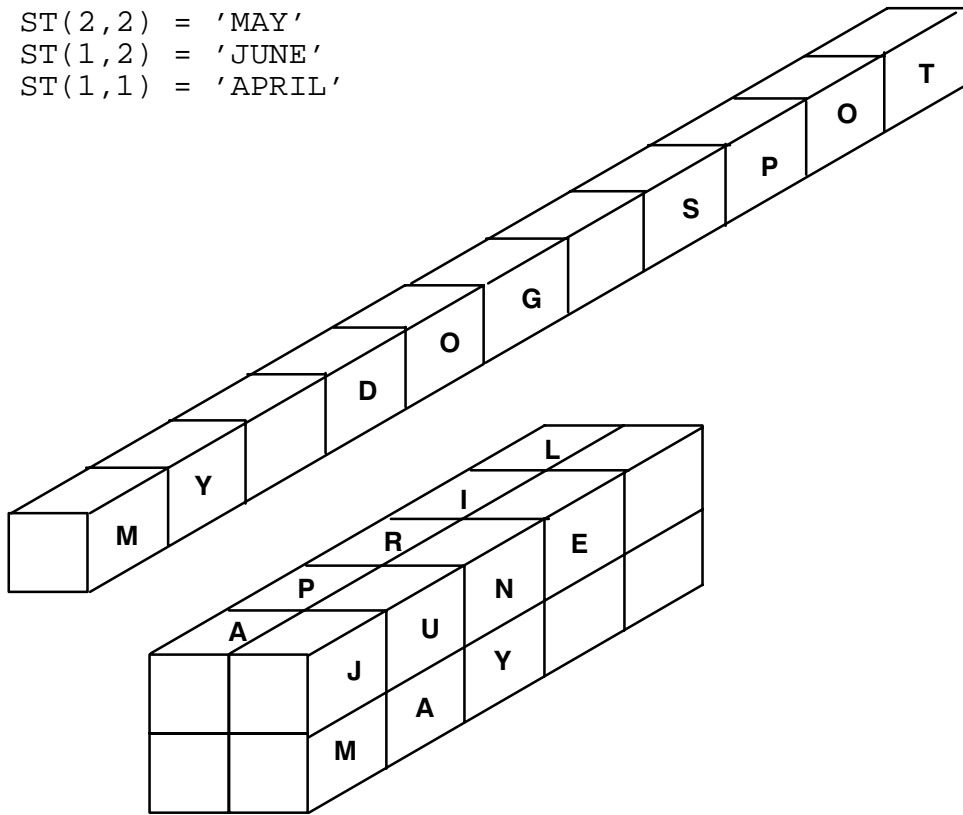


Figure 2–4 String Variable and Two Dimensional Arrays

In GRIP you may express a string in three ways:

1. String Literal

A sequence of characters enclosed in single quotes, with a maximum of 132 characters.

Example

```
MESSG/ 'THIS IS A STRING LITERAL'
```

displays the message on the dialog area.

2. String Variable

You must declare a string by using the `STRING/` declaration before you can use it. You may assign a value to a string variable by using the general format:

```
STRING VARIABLE = 'STRING LITERAL'
```

If it is not assigned, the content of the string is assumed to be null.

Example

```
STRING/A(5)  
.  
.  
.  
A = 'LINES'  
NOTE/1,1,A
```

The variable `A` is declared a string, then assigned the value `'LINES'`.

The variable `A` can then be used in place of the string literal `'LINES'` throughout the rest of the program.



3. String Expression

You may use a string expression to set up a combination of string literals, string variables and string-valued functions. Strings may be added (concatenated) using the plus sign (+).

Example

```
STRING/A(6),B(16)
.
.
.
A = 'MY DOG'
B = A + BLSTR(1) + 'HAS FLEAS'
```

results in the string:

```
B = 'MY DOG HAS FLEAS'
```

BLSTR(1) is a string function which generates a one character string of blanks (' '). You will learn about string functions later in this student guide.



Initializing and Setting Variables

The DATA statement initializes the values of numerical and string variables. The string and numerical variables included in a DATA/ statement must first be declared using the STRING and NUMBER declarations. The initialization occurs the first time a subroutine is executed but does not occur on subsequent calls to that routine.

The data item list may consist of items of the following types:

Form	Statement Example
string, 'string'	DATA/STR, 'ENTER'
string array element, 'string'	STRING/MES(2,2,6) DATA/MES(1,2), 'VALUES'
num. var, value	DATA/X, 5
num. array, list	NUMBER/A(2) DATA/A, -1.2, 8
num. array element, value	NUMBER/A(6) DATA/A(3), 8
num. array, value	NUMBER/B(2) DATA/B, 3

The values assigned to numerical variables and arrays must be constants and the subscripts used to specify array elements must also be constants. The variable length subrange operation cannot be used in a DATA/ statement.

Example

The following example of statements assign 'ENTER' to STR, -1, 2, 5 to A(1), A(2), A(3) respectively, and 3 to B(1).

```
NUMBER/A(3), B(10)
STRING/STR(50)
DATA/STR, 'ENTER', A, -1, 2, 5, B, 3
```

To check the amount of numbers, strings and entities that have been declared, refer to the last line on the Unigraphics GRIP compilation screen. The number after the word DATA/ gives the amount of numbers, strings, and entities which have been declared.

Assigning Values to String Variables

You have three ways to assign a value to a string variable:

1. Direct Assignment

```
STRING/str1(20)
str1='NOTE:LINES'
```

Advantages: You can use anywhere in the program. STRING/ must be before the first executable line.

Disadvantages: You can only assign one variable at a time. Occurs during Run which slows processing

2. Data Statement Assignment

```
NUMBER/NUM
STRING/STR(10),STR2(35)
DATA/STR,'THE NUMBER',NUM,0,$
STR2,' IS USED TO CALCULATE DISTANCE.'
```

Advantages: Assigns values during compile.

Disadvantages: Must occur directly after the declaration statements.

3. Null Assignment

```
ENTITY/ nm_note
STRING/ str(30)

str = 'Pat Smith'
nm_note = NOTE/ 1, 5, str
str = &NULSTR
```

&NULSTR is a GPA constant that can be used to assign a null value to a string variable.



Assigning Values to Numeric Variables

Numeric variables declared in the “NUMBER/” statement are all initially zero.

Example

```
NUMBER/A(3)
```

All three parts of array A are set to zero.

You have three ways to assign a value to a numerical variable:

1. Direct Assignment

```
NUMBER/A(3)
A(1)=7.125
A(2)=0
A(3)=1
```

Advantages: Can be anywhere in the program.

Disadvantages: Can only assign one variable at a time. Occurs during Run which results in slower processing.

2. Data Statement Assignment

```
NUMBER/A(3),B(3)
DATA/A(1),7.125,A(2),0,A(3),1,B,5,9.1,29.6
```

Advantages: Assigns value during Compile which results in faster processing.

Can assign values to entire array two ways.
The assignment to B results:

```
B(1)=5
B(2)=9.1
B(3)=29.6
```

Disadvantages: Must occur directly after the declaration statements.

Numerical values may be in exponential notation:

“Mantissa” E “exponent”

The mantissa can be a sign and up to 15 digits.
The exponent can be from -79 to $+75$.



3. Null Assignment

```
ENTITY/ nm_note
STRING/ str(30)
NUMBER/ loc_note(2)
```

```
str = 'Pat Smith'
nm_note = NOTE/ loc_note, str
loc_note(1) = 0
loc_note(2) = 0
```

Numbers can never really be “null” in GRIP. However, a zero value can be used to assign a null value to a number variable.

Example

```
NUMBER/ A(3)
DATA/ A(1), 32.3E-3    $$ A(1) = .0323
A(2) = -.1234E6    $$ A(2) = -123400
```

All numbers in GRIP are real.

Assigning Values to Entity Variables

There are three ways of assigning values to entity variables. The DATA/ statement is never used to initialize the value of entity variables.

1. Creating an Entity

```
ENTITY/ part_edge(4)

part_edge(1) = LINE/ 0, 0, 0, 1, 2, 0
```

2. Reading the Identifier of an Entity

```
ENTITY/ nam_note

nam_note = &ENAME(1, 'drafter_name', IFERR, LBL050:)
```

The EDA &ENAME is used to examine the Unigraphics data for the first occurrence of the entity with the attribute name ‘drafter_name’.

3. Null Assignment

```
ENTITY/ part_edge(4)
```

```
part_edge(1) = LINE/ 0, 0, 0, 1, 2, 0
```

```
part_edge(1) = &NULENT
```

&NULENT is a GPA constant that can be used to assign a null value to an entity variable.

Subrange Operators

When you are using arrays, many times it is desirable to refer to portions of an array rather than the entire array. The subrange operator “..” addresses a range of elements in an array.

You have three methods of expressing the upper and lower bounds of a subrange:

1. Constant Subrange:

The bounds are constants or are expressions that reduce to constants.

Example

In the example below, the bounds are from 1 to 3, and from 12 to 18

```
N(1..3), N(4*3..18)
```

2. Fixed Array Subrange:

The lower bound is a variable and the upper bound is a constant added to the variable.

Example

In the example below the upper bounds of J and K ranges are J+7 and k+3 respectively.

```
N(J..J+7), N(I,J,K..K+3)
```

3. Variable Array Subrange:

The lower and upper bounds may be expressed as any valid arithmetic expressions.

Example

In the example below, $ABS(X+3)$ expresses the lower bound and $I*2$ expresses the upper bound.

$N(ABS(X+3) .. I*2)$

Rules for Using Subrange Operators

- When subranges are used at run time, the difference between the first and last number in the subrange must be positive. If this is not true, a run time error occurs, and you will be informed that the difference between subrange operators cannot be negative.
- You may use constant and fixed array subranges anywhere in a GRIP program where an entire unsubscripted array can be used (except in a data statement).
- You may only use variable array subranges in the following GRIP functions: BLANK, BOUND, CLINE, DELETE, DRAW, GROUP, HATCH, IDENT, MASK, and UNBLNK.



Example Array Subrange Usage

```
ENTITY/E1(50),E2(10,10)
NUMBER/N1(200),N2(10,10),N3(5,10)
```

Table 2–2 Array Assignments

Assignment	Validity
N1(51..150)=N2	Valid assignment of N1 because 150 is less than 200. Same number of array elements on either side of the equals sign.
N3=N1(I..J)	Valid. No compilation error. At run time, error checks will be made on array bounds and the subrange.
N1(6..10)=N1(1..5)	Valid assignment.
POINT/N1(20..22)	Valid.
POINT/N1(I..I+2)	Valid.
POINT/N1(J..K)	Invalid to use variable array subrange in a POINT definition.
HATCH/E1(J..K)	Valid. Variable subrange can be used with HATCH statement.
GROUP/E2(1..I,1..J)	Invalid. Only one array subrange can be used in a multidimensional array.



Detect Undeclared Variables

DESCRIPTION

The **GRIPSW** statement is used for compiler directives. If used with the minor word directive **DECLRV**, it prevents the program from compiling if the program contains undeclared simple variables. This ensures that all variables are declared before they are used in the program. This reinforces the data assignment to the correct variable types.

During compilation, if the compiler comes across an undefined simple variable, it will generate error number 62, **UNDEFINED VARIABLE**.

You can place the **GRIPSW** statement anywhere in the declaration section, but it must come before the first executable statement. For example, this declaration:

```
NUMBER/a,b
GRIPSW/DECLRV
STRING/STR(2,30),NAME(6)
.
.
a = 10
.
.
```

would have the same result as this:

```
GRIPSW/DECLRV
NUMBER/a,b
STRING/STR(2,30),NAME(6)
.
.
a = 10
.
.
```

FORMAT	GRIPSW/DECLRV
REQUIRED INPUT	DECLRV
OPTIONAL INPUT	NONE
PARAMETERS	DECLRV Minor word which detects undeclared variables.



EXAMPLE

DESCRIPTION This example demonstrates the usage of the **GRIPSW/DECLRV** statement in a program.

```

NUMBER/A,B
STRING/STR(2,30),NAME(6)
GRIPSW/DECLRV
.
.
.
A = 10
NUM = A + B
PRINT/NUM
.
.
.
HALT

```

AFTER COMPILATION:

```

1 NUMBER/A,B
2 STRING/STR(2,30),NAME(6)
3 GRIPSW/DECLRV
.
.
.
27 A = 10
28 NUM = A + B
*
```

Error no. 62 in line no. 28: Undefined variable

```

29 PRINT/NUM
*
```

Error no. 62 in line no. 29: Undefined variable

```

.
.
.
45 HALT

```

Exercises

Complete the following exercises to familiarize yourself with the language components of GRIP. Refer to the prior pages in this manual, or to the *GRIP Programming Manual* for further information.

Exercise 2–1

Write declarations and/or initialization statements for the data as described.

1. An engineer wants you to write a program that will parametrically construct the outer edge of a part that will be later manufactured using a lathe. Upon further questioning, you determine that the maximum number of entities expected for the part edge is 200.
2. You are designing an application that will store the full path name to a part file. A full path name to a UNIX file can contain up to 132 characters.
3. You are developing an application that will store the following data with the following default values that will be requested by the user:

Numeric information

length	10.2
width	8
height	.5

Character string information

part name	PLATE
designer name	XX XXXXXXXXXXXX
date	DD MMM YYYY



Exercise 2–2

The following statements are a scrambled GRIP program. Using the program structure outline on page 2–2, rewrite the statements in their correct order.

```
ctr_pt = POINT/ xc, yc, zc  
HALT  
DATA/ strt_coord, 10, 20  
xc = 5  
NUMBER/ strt_coord(3), xc, yc, zc  
yc = 10  
ENTITY/ ctr_pt, strt_pt  
strt_pt = POINT/ strt_coord(1..3)
```



Two Dimensional Commands

Lesson 3

Objectives

- Demonstrate an understanding of two dimensional modeling commands including points and lines.
- Become familiar with the way the options of the processing commands are documented.



Positional Modifiers

A positional modifier helps the system determine which of several geometric possibilities will be used. In GRIP documentation, PMOD2 is used to represent positional modifiers for two dimensional objects; PMOD3 is used to represent positional modifiers for three dimensional objects. Positional modifiers identify quadrant locations or end positions of existing objects. The six PMOD Minor words are XSMALL, YSMALL, ZSMALL, XLARGE, YLARGE, and ZLARGE.

Two-Dimensions – PMOD2

For two dimensional objects, use one of the PMOD2 modifiers or the corresponding number listed below. In PMOD2, the ZSMALL and ZLARGE positional modifiers are not allowed.

Minor Word	Corresponding Number
XSMALL	1
YSMALL	2
XLARGE	4
YLARGE	5

3

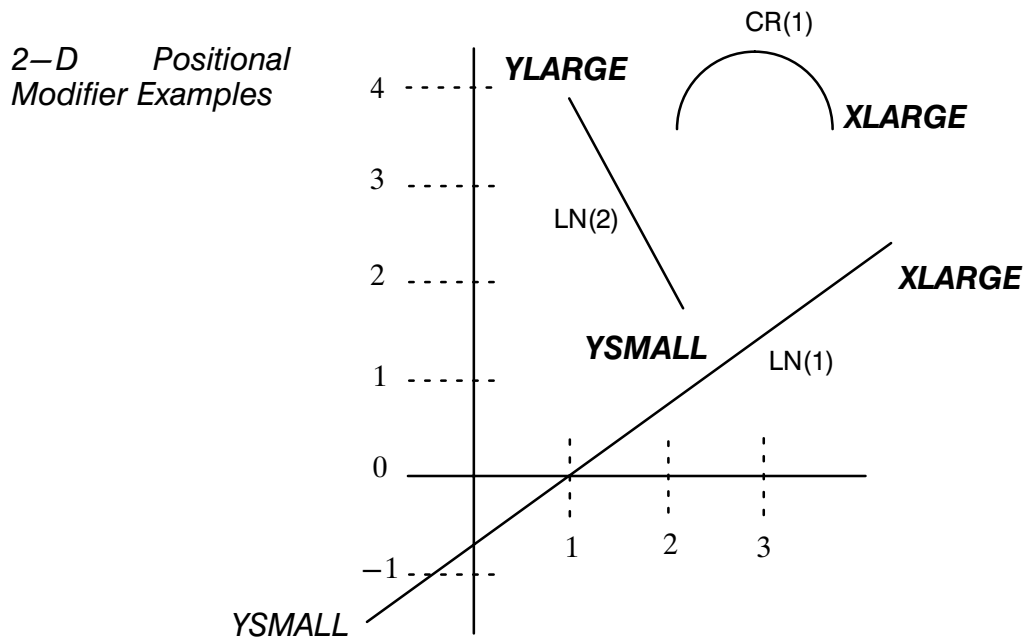


Figure 3–1 2-D Positional Modifiers

Three-Dimensions – PMOD3

For three dimensional objects, use the PMOD3 modifiers listed below, or the corresponding numbers.

Minor Word	Corresponding Number
XSMALL	1
YSMALL	2
ZSMALL	3
XLARGE	4
YLARGE	5
ZLARGE	6

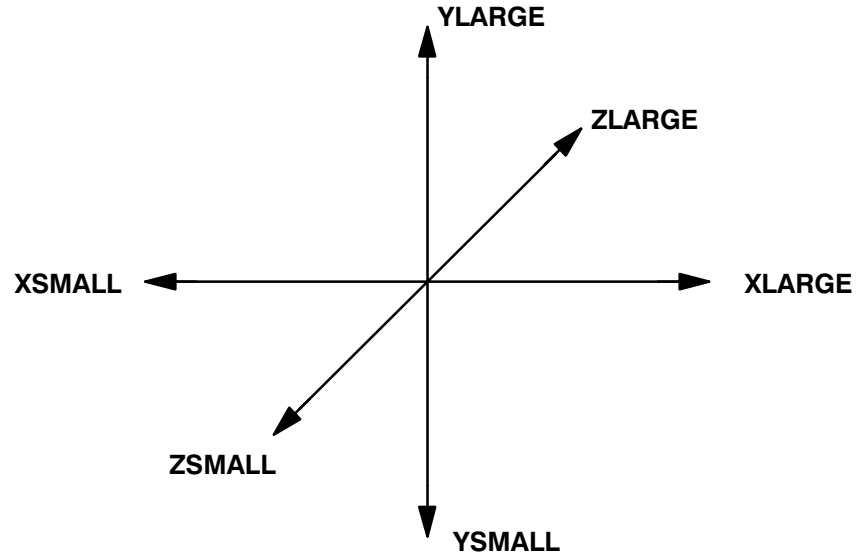


Figure 3–2 3-D Positional Modifiers (Top View)

Point

The **POINT** statement creates points using several different methods. You can create a point by specifying its work coordinates or by using geometric construction techniques to create the point relative to other objects.

Table 3–1 Point Command Statements

Function	Format
Coordinates	POINT/x,y[,z]
Center of a Circle	POINT/CENTER, <u>circle</u>
End Point	POINT/ENDOF, “P <u>MOD3</u> ”, <u>obj</u>
Intersection Point	POINT/[{“P <u>MOD2</u> ” <u>point</u> },] INTOF, <u>obj1</u> , <u>obj2</u> [,IFERR,label:]

Point: Coordinates

Synopsis

obj = POINT/x,y[,z]

Description

Creates a point by specifying its coordinates in XC, YC, and ZC. The ZC value is optional. If it is omitted, the point will lie on the current work plane.

Parameters

x,y,z

The X,Y, and Z coordinates of the work coordinate system. If the optional Z coordinate is omitted, the Z coordinate of the point will be equal to zero.

Example

This example demonstrates the creation of several points by specifying their coordinates.

Declarations

ENTITY/PT1,PT2,PT3

Point Definition

PT1=POINT/0,0
PT2=POINT/1,0,1
PT3=POINT/1,0

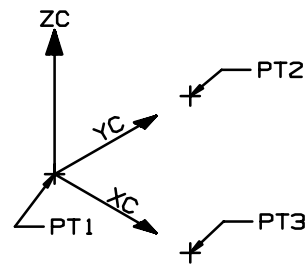


Figure 3–3 Points defined by coordinate values

Point: Center of a Circle

Synopsis **obj = POINT/CENTER,circle**

Description Creates a point at the center of a previously defined circle.

Parameters **CENTER**
Minor word that indicates that the point defined will be the center of a circle.

circle
An existing circle.

Example This example demonstrates the creation of a point at the center of a previously defined circle.

Declarations ENTITY/CR1,PT1

Geometry Definition CR1=CIRCLE/0,0,1

Point Definition PT1=POINT/CENTER,CR1

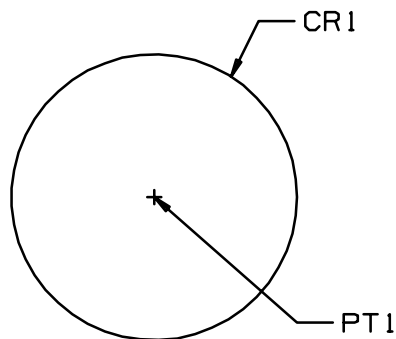


Figure 3-4 A point at the center of a circle

Point: End Point

<i>Synopsis</i>	<u>obj</u> = POINT/ENDOF,"PMOD3",<u>obj</u>
<i>Description</i>	Creates a point at the end of a previously defined object. A positional modifier is used to define which end of the object is desired.
<i>Parameters</i>	<p>ENDOF Minor word that indicates that the point defined will assume the coordinate values at the end of an object.</p> <p>PMOD3 A three dimensional positional modifier which indicates which end of the object will be used. The direction will be in reference to the work coordinate system.</p> <p><u>obj</u> An existing object which may be any one of the following curves: line, circle, conic, or spline</p>
<i>Example</i>	This example demonstrates the creation of several points at the ends of a line and a circle.
<i>Declarations</i>	ENTITY/LN1,CR1,PT1,PT2,PT3,PT4
<i>Geometry Definition</i>	LN1=LINE/-2,-1,0,1 CR1=CIRCLE/1,0,1,START,45,END,210
<i>Point Definition</i>	PT1=POINT/ENDOF,XSMALL,LN1 PT2=POINT/ENDOF,YLARGE,LN1 PT3=POINT/ENDOF,XLARGE,CR1 PT4=POINT/ENDOF,YSMALL,CR1

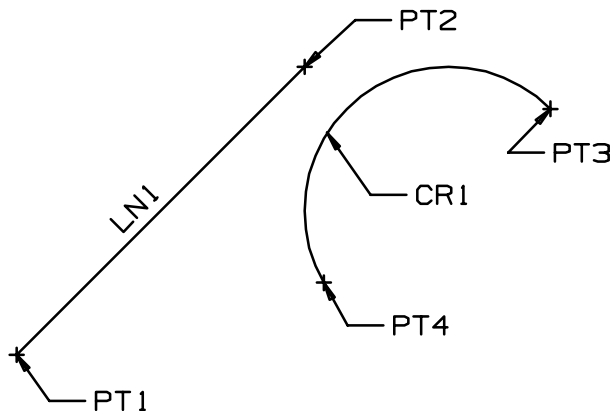


Figure 3-5 A point at the end of an object

Point: Intersection Point

Synopsis

**obj = POINT/[{"PMOD2" | point},]INTOF,obj1,obj2
[,IFERR,label:]**

Description

Creates a point at the intersection of two previously defined objects. A reference point may be specified to indicate the desired point in the case of multiple intersections of the two objects. The intersection is determined by looking down the Z-axis of the WCS.

Parameters

PMOD2

A two dimensional positional modifier which may be used to indicate which intersection will be used when multiple intersections exist. The direction will be in reference to the work coordinate system.

point

Reference point closest to the desired intersection which may be used instead of the positional modifier, when both objects are curves. If the desired point is the intersection of a curve and a surface or plane, a previously defined point must be used.

NOTE

A positional modifier is used to assign a reference point. This calculated reference point may not yield the expected results.

INTOF

Minor word that indicates that the point defined will be a unique intersection of two objects.

obj1,obj2

Two existing objects which are not parallel in the work view. If one of the objects is a surface or plane, it must be obj2.

IFERR,label:

Specifies label to which program execution will jump if no intersection is found.

<i>Example</i>	This example demonstrates the creation of several intersection points between two sets of circles.
<i>Declarations</i>	ENTITY/CR1,CR2,CR3,CR4,PT0,PT1,PT2,PT3,PT4
<i>Geometry Definition</i>	CR1=CIRCLE/-1.25,0,.5 CR2=CIRCLE/-.5,0,.5 CR3=CIRCLE/1,.375,.5 CR4=CIRCLE/1,-.375,.5 PT0=POINT/1.5,0
<i>Point Definition</i>	PT1=POINT/YLARGE,INTOF,CR1,CR2 PT2=POINT/YSMALL,INTOF,CR1,CR2 PT3=POINT/XSMALL,INTOF,CR3,CR4 PT4=POINT/PT0,INTOF,CR3,CR4

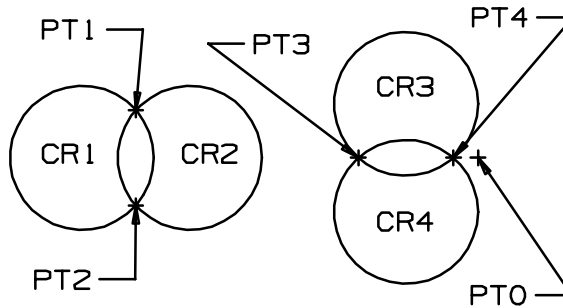


Figure 3–6 Intersection points of two curves



Point: EDA*Synopsis***&POINT(obj)***Description*

Accesses the location of an existing point. The coordinates in X, Y, and Z of the work coordinate system may be extracted and/or edited.

Characteristics

Read/Write Number Three Position Numerical Array

Example

This example demonstrates the use of the **&POINT** EDA to edit the location of a point. A point is created and its coordinates displayed in a menu using the **&POINT** EDA in a read mode. Once the user enters the new coordinates, the point is edited by using the **&POINT** EDA in a write mode.

```
ENTITY/ ref_point
NUMBER/ ref_coord(3), resp
$$
$$ Create point
$$
ref_point = POINT/ 3, 0, 0
$$
$$ Extract coordinates of the point
$$
ref_coord = &POINT(ref_point)
$$
$$ Change the values of the coordinates
$$
ref_coord(1) = ref_coord(1)+1.5
ref_coord(2) = ref_coord(2)+2
ref_coord(3) = ref_coord(3)+1
$$
$$ Edit the point using the new coordinates
$$
&POINT(ref_point) = ref_coord
CANCEL:
HALT
```

Line

The **LINE** statement creates lines using several different methods. You can define a line between end points or use geometric construction principles using other existing geometry.

Table 3–2 Line Command Statements

Type	Format
Between Two Specified Points	LINE/x1,y1[,z1],x2,y2[,z2]
Between Two Existing Points	LINE/ <u>point1</u> , <u>point2</u>
Parallel at a Distance	LINE/PARLEL, <u>line</u> ,"PMOD3",offset
Thru a Point, at an Angle	LINE/ <u>point</u> ,ATANGL,angle
Point, Tangent to a Curve	LINE/ <u>point1</u> ,{LEFT RIGHT <u>point2</u> },TANTO, <u>curve</u>
Thru a Point, Parallel/Perpendicular to a Line	LINE/ <u>point</u> ,{PARLEL PERPTO}, <u>line</u>

Line: Between Two Specified Points (Coordinates)

Synopsis **obj = LINE/x1,y1[,z1],x2,y2[,z2]**

Description Creates a line by specifying the coordinates of the end points of the line.

The Z value at both positions is optional, therefore, if only the X and Y coordinates of an end point is specified, the current depth setting will be assumed for the Z coordinate. If both Z values are omitted, the line will lie in a plane parallel to the X-Y plane of the work coordinate system.

Parameters **x1,y1,z1,x2,y2,z2**

The coordinate values of two points between which the line will be created. The start point of the line will be at coordinates x1,y1,z1 and the end will be at coordinates x2,y2,z2.

Example

This example demonstrates the creation of two lines by specifying the coordinates of their end points.

LN1 lies on the current work plane because only X and Y values were specified.

LN2 is defined in space because X, Y, and Z values were specified.

Declarations ENTITY/LN1, LN2

Line Definition LN1=LINE/-1,-1,1, 1
LN2=LINE/-1,1,1,1,-1,-1

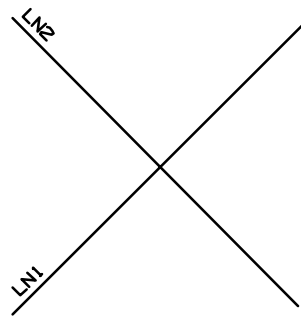


Figure 3–7 Top view of lines created with two and three dimensional values



Line: Between Two Existing Points

<i>Synopsis</i>	<u>obj = LINE/point1,point2</u>
<i>Description</i>	Creates a line between two previously defined points. A point array may be used to define the two points.
<i>Parameters</i>	<u>point1, point2</u> Two previously defined points between which the line is created. The start point of the line is point1 and the end point is point 2.
<i>Example</i>	This example demonstrates the creation of a line between two previously defined points.
<i>Declarations</i>	ENTITY/PT1,PT2,LN1
<i>Geometry Definition</i>	PT1=POINT/-1,-1 PT2=POINT/PT1,DELTA,2,2,0
<i>Line Definition</i>	LN1=LINE/PT1,PT2

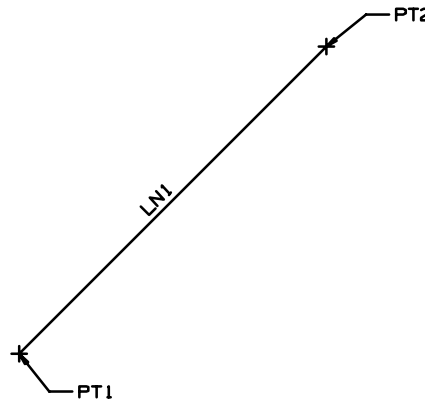


Figure 3–8 A line between two points.

NOTE

Since PT1 and PT2 were defined with two dimensional values the line LN1 will lie in the X-Y plane of the work coordinate system, if the work coordinate system is not changed between the point definitions and the line creation.

Line: Parallel at a Distance

Synopsis **obj = LINE/PARLEL,line,"PMOD3",offset**

Description Creates a line parallel to an existing line at a specified distance. A positional modifier is used to indicate which side of the existing line the new line will be created on.

If XLARGE, XSMALL, YLARGE, YSMALL are specified, the distance will be offset in the current work plane.

If ZLARGE or ZSMALL is specified the distance will be offset parallel to the current Z axis.

Parameters **PARLEL**
Minor word that indicates that the new line will be parallel to an existing line.

line
An existing line to which the new line will be parallel.

PMOD3
The positional modifier PMOD3 establishes the relationship of the new line to the existing line, with respect to the work coordinate system.

offset
The distance from the existing line to the new line.

NOTE The length of the new line will be the same as the existing line.



<i>Example</i>	This example demonstrates the creation of a line parallel to another existing line at a specified distance.
<i>Declarations</i>	ENTITY/LN1, LN2, LN3
<i>Geometry Definition</i>	LN1=LINE/-1,0,1,0
<i>Line Definition</i>	LN2=LINE/PARLEL, LN1, YLARGE, 1 LN3=LINE/PARLEL, LN1, YSMALL, 1

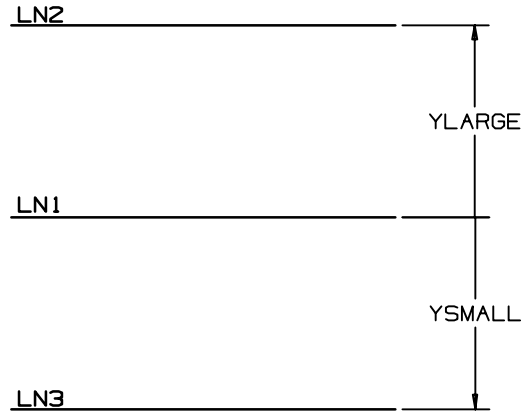


Figure 3–9 Lines parallel to an existing line at a given distance

Line: Through a Point, At an Angle

Synopsis

obj = LINE/point,ATANGL,angle

Description

Creates a line thru a previously created point at a specified angle. The angle is measured counterclockwise from the current positive X axis. The line is created, and the angle is measured in a plane containing the point which is parallel to the current work plane.

Parameters

point

A previously defined point through which the line will run.

ATANGL

Minor word that indicates that the angle of the line will be specified.

angle

The angle of the line in degrees, which is measured from the positive X axis of the work coordinate system to the line. A positive angle will rotate the line counterclockwise about the specified point and a negative angle will rotate the line clockwise.

Example

This example demonstrates the creation of two lines that pass thru PT1 at a specified angle.

Declarations

ENTITY/PT1,LN1,LN2

Geometry Definition

PT1=POINT/0,0

Line Definition

LN1=LINE/PT1,ATANGL,45
LN2=LINE/PT1,ATANGL,-30

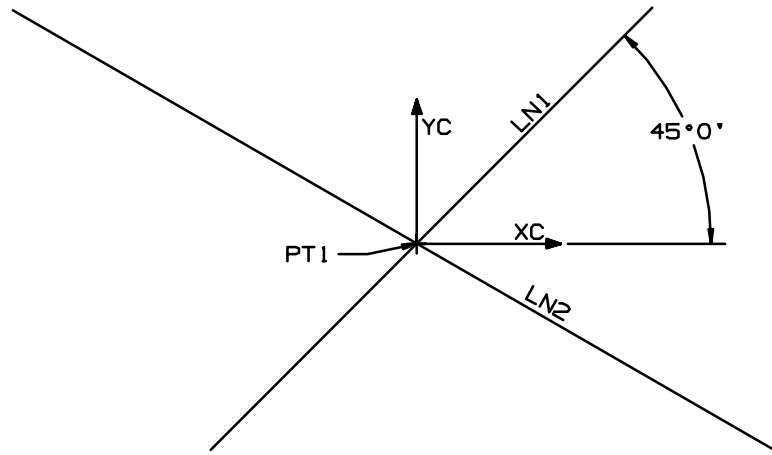


Figure 3–10 Line through a point at an absolute angle



Line: Point, Tangent to a Curve

Synopsis **obj = LINE/point1,{LEFT|RIGHT|point2},TANTO,curve**

Description Creates a line from specified point tangent to a previously defined curve. You may specify either a point or use the minor words RIGHT or LEFT to define which side of the curve the line should be tangent to. If you use RIGHT or LEFT, position yourself at the point looking at the curve.

Parameters **point1**
A previously defined point from which the line will start.

LEFT,RIGHT
Positional modifiers which may be used to indicate which side of the existing curve the line will be tangent to, as viewed from the point to the curve. LEFT and RIGHT are not recommended for splines.

point2
A previously defined point may be used in place of the positional modifier to approximate the point of tangency.

TANTO
Minor word that indicates that the new line will be tangent to an existing curve.

curve
An existing curve to which the new line will be tangent.



Example

This example demonstrates the creation of two lines from PT1 tangent to a previously defined circle.

LN1 is defined using PT2 to specify which side of the circle it should be tangent to.

LN2 is defined using a the minor word RIGHT.

Declarations

ENTITY/PT1,PT2,CR1,LN1,LN2

Geometry Definition

PT1=POINT/-1,0

PT2=POINT/1,1

CR1=CIRCLE/1,0,.5

Line Definition

LN1=LINE/PT1,PT2,TANTO,CR1

LN2=LINE/PT1,RIGHT,TANTO,CR1

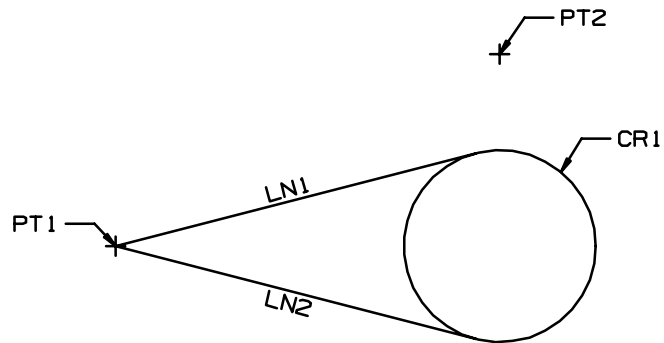


Figure 3–11 Line through a point and tangent to a curve

Line: Through a Point, Parallel/Perpendicular to a Line

Synopsis

obj = LINE/point, {PARLEL | PERPTO}, line

Description

Creates a line thru a previously defined point which is either parallel or perpendicular to a specified line. The line will be created either parallel or perpendicular to the specified curve depending upon which minor word (PARLEL or PERPTO) is used.

Parameters

point

A previously defined point from which the perpendicular line will start or through which the parallel line will run.

PARLEL

Minor word that indicates that the new line will be parallel to an existing line.

PERPTO

Minor word that indicates that the new line will be perpendicular to an existing line.

line

An existing line to which the new line will be parallel or perpendicular.

NOTE

If PARLEL is used, the new line will be the same length as the existing line if the projected point falls between the existing line endpoints. Otherwise, the end closest to the point will be extended to the point.



Example This example demonstrates the creation of several lines which are parallel or perpendicular to a line (LN1) and pass thru a previously defined point.

Declarations ENTITY/PT1,PT2,LN1,LN2,LN3,LN4

Geometry Definition
 PT1=POINT/0,-.5
 PT2=POINT/0,.5
 LN1=LINE/-1,-.5,0,.5

Line Definition
 LN2=LINE/PT1,PARLEL,LN1
 LN3=LINE/PT1,PERPTO,LN1
 LN4=LINE/PT2,PERPTO,LN1

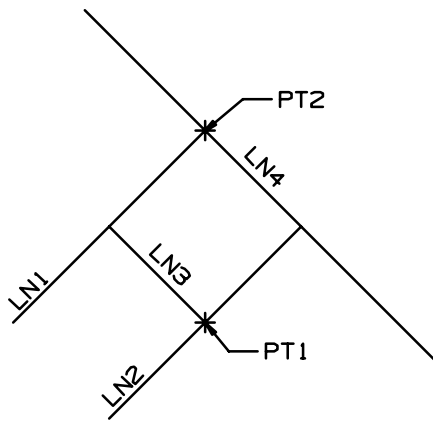


Figure 3–12 Lines through points, parallel or perpendicular to a line



Line EDA's

There are several EDA's which access the geometric properties of a line. Some of the EDA's are READ/WRITE, which means that the properties can be accessed and edited. READ ONLY EDA's can be accessed to extract the geometric properties only.

Geometric Property	EDA Symbol	Access Type	Data Type
Start point	&SPOINT(<u>obj</u>)	R/W	N(3)
End point	&EPOINT(<u>obj</u>)	R/W	N(3)
Length	&LENGTH(<u>obj</u>)	RO	Number



Example

This example demonstrates the use of the line EDA's in conjunction with the **&POINT** EDA to trim a line.

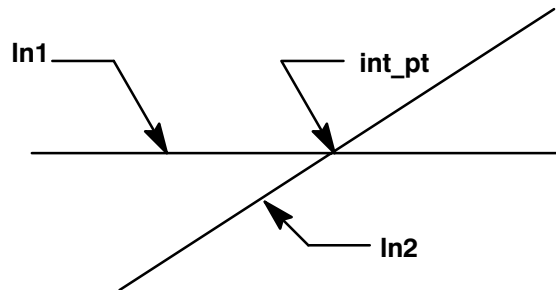


Figure 3–13 Untrimmed: the objective is to trim ln2 to the intersection between ln1 and ln2

```
int_pt = POINT/ INTOF, ln1, ln2
scoord = &SPOINT(ln2)
ecoord = &EPOINT(ln2)
$$
$$ Evaluate the coordinates of ln2 to determine
$$ whether the starting coordinate or ending
$$ coordinate should be edited
$$
IFTHEN/ scoord(1) < ecoord(1)
    &EPOINT(ln2) = &POINT(int_pt)
ELSE/
    &SPOINT(ln2) = &POINT(int_pt)
ENDIF
DELETE/ int_pt
```

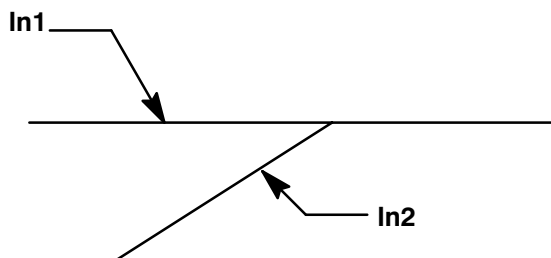


Figure 3–14 Trimmed: ln2 is trimmed to the intersection between ln1 and ln2

Activity *The L-shape*

This first activity will let you begin programming in GRIP. Look at the diagram of the L-Shape on the next page. Write a program to create the figure. Create a new file using your initials (***-lshape.grs) and follow these directions:

- Use line definitions only. Do not use any point definitions.
- Write the program so that the X and Y coordinates for the lowest, leftmost corner of the figure are the only coordinates that will be given.
- The following number of variables and their accompanying values are the only ones to be used.

Variable	Value	Definition
x	0	X coordinate of lowest, leftmost corner of figure.
y	-3	Y coordinate of lowest, leftmost corner of figure.
bw	1	Bottom width of figure
tw	2	Top width of figure
tht	6	Total height of figure
tpht	2	Top height of figure

Notice that the variables are numeric. This means you cannot use them as object names. The variables represent the lengths of the lines.

TIP This project will need declaration and initialization statements:

```
ENTITY/  
NUMBER/  
DATA/
```

six LINE/ statements such as:

```
ln(1) = LINE/x, y, x, (y + tht)
```

and the program termination statement:

```
HALT
```

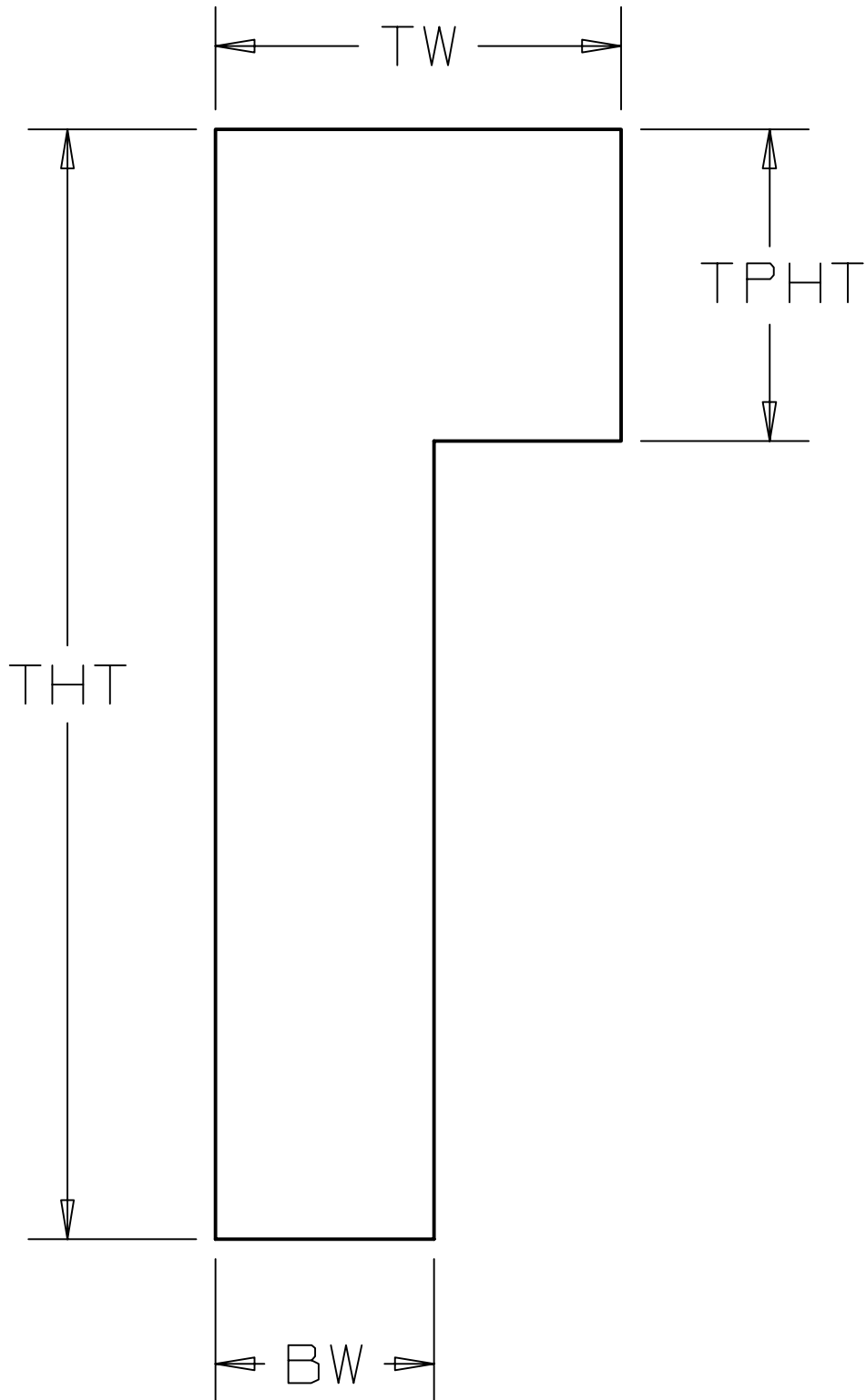



Figure 3–15 The L–Shape

(This Page Intentionally Left Blank)



More Two Dimensional Commands

Lesson 4

Objectives

- Demonstrate an understanding of two dimensional circle commands.
- Become familiar with the way the options of the processing commands are documented.
- Demonstrate knowledge of the Entity Data Access Symbols (EDA's) used when accessing and altering information about geometric objects.



Circle

A **Circle** is created in a counterclockwise direction from the positive X axis of the work coordinate system. The start and end angles of the arc may be specified in some of the circle definitions. If the optional start and end angles are omitted in the definition statement, a full circle will be created.

Table 4–1 Circle Command Statements

Function	Format
Center Coordinates, Radius	CIRCLE/ x, y, [z,] r [,START, start angle, END, end angle]
Center Point, Radius	CIRCLE/ CENTER, <u>point</u> , RADIUS, r [,START, start angle, END, end angle]
Center Point, Tangent to Line	CIRCLE/ CENTER, <u>point</u> , TANTO, <u>line</u> , [,START, start angle, END, end angle]
Center Point, Point on Arc	CIRCLE/ CENTER, <u>point1</u> , <u>point2</u> , [,START, start angle, END, end angle]
Through Three Points	CIRCLE/ <u>point1</u> , <u>point2</u> , <u>point3</u>



Circle: Center Coordinates, Radius

Synopsis

obj = CIRCLE/x,y,[z],r[,START,start angle,END,end angle]

Description

Creates a circle by defining the X, Y, and Z coordinates in work coordinates and the radius. The Z coordinate is optional. The plane of the arc will be parallel to the plane of the current WCS.

Parameters

x,y,[z]

The coordinate values of the circle center point. If the optional Z coordinate is omitted, the center point will assume the Z value of the currently set depth.

r

A positive value which establishes the circle radius.

START

The optional parameter START indicates that the starting angle for the arc will be specified.

start angle

The starting angle of the arc as measured from the positive X axis of the work coordinate system.

END

The optional parameter END indicates that the ending angle for the arc will be specified.

end angle

The ending angle of the arc as measured from the positive X axis of the work coordinate system.

Example

This example shows the creation of a full and partial circle. The partial circle demonstrates the use of the START and END minor words.

Declarations

ENTITY/CR1,CR2

CR1=CIRCLE/1,0,1,.5

CR2=CIRCLE/3,0,.5,START,30,END,270

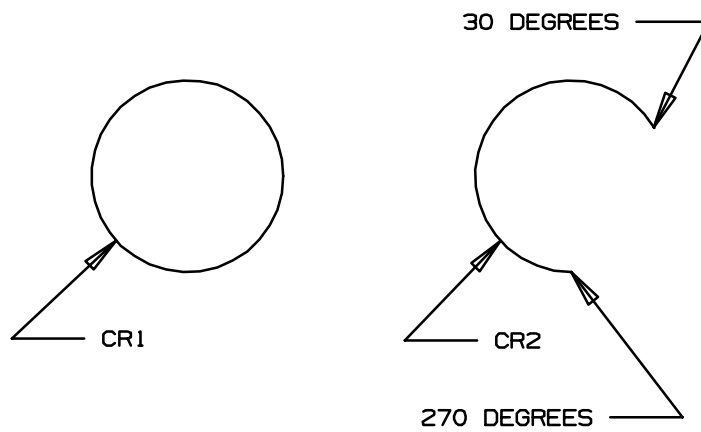


Figure 4–1 Arcs with specified center point coordinates and radius



Circle: Center Point, Radius

Synopsis

obj = CIRCLE/CENTER,point,RADIUS,r
[,START,start angle,END,end angle]

Description

Defines a circle with a specified radius using a previously defined point as its center. You may also define start and end angles for the circle.

Parameters

CENTER

Minor word that indicates that the center of the circle is an existing point.

point

A previously defined point which establishes the center of the circle.

RADIUS

Minor word that indicates that the radius of the circle will be specified.

r

A positive value which establishes the circle radius.

START

Indicates that the starting angle for the arc will be specified.

start angle

The starting angle of the arc as measured from the positive X axis of the work coordinate system.

END

Indicates that the ending angle for the arc will be specified.

end angle

The ending angle of the arc as measured from the positive X axis of the work coordinate system.



Example The following example shows the creation of a circle using a center point and radius.

Declarations ENTITY/PT1,CR1

Geometry Definition PT1=POINT/0,0

Circle Definition CR1=CIRCLE/CENTER,PT1,RADIUS,1

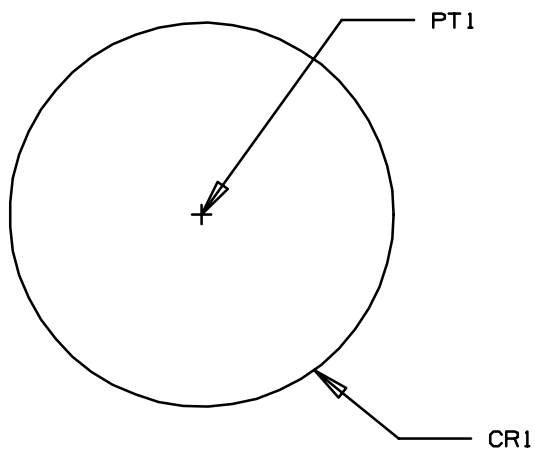


Figure 4–2 Circle with a specified center point and radius



Circle: Center Point, Tangent to a Line

Synopsis

obj = CIRCLE/CENTER,point,TANTO,line
[,START,start angle, END,end angle]

Description

Creates a circle which is tangent to a specified line using a previously defined center point. The arc lies in the plane containing the center point and the line. You may also define start and end angles for the circle.

Parameters

CENTER

Minor word that indicates that the center of the circle is an existing point.

point

A previously defined point which establishes the center of the circle.

TANTO

Minor word that indicates that the circle will be tangent to an existing object.

line

An existing line to which the circle will be tangent .

START

The optional parameter START indicates that the starting angle for the arc will be specified.

start angle

The starting angle of the arc as measured from the positive X axis of the work coordinate system.

END

The optional parameter END indicates that the ending angle for the arc will be specified.

end angle

The ending angle of the arc as measured from the positive X axis of the work coordinate system.

NOTE If the plane defined by the line and center point are not parallel to the WCS, the system ignores the start and end angles and creates a full circle.

Example The following example shows how to create a circle with a center point, tangent to a line.

ENTITY/PT1,LN1,CR1

PT1=POINT/0,0

LN1=LINE/2,0,0,2

CR1=CIRCLE/CENTER,PT1,TANTO,LN1,START,0,END,90

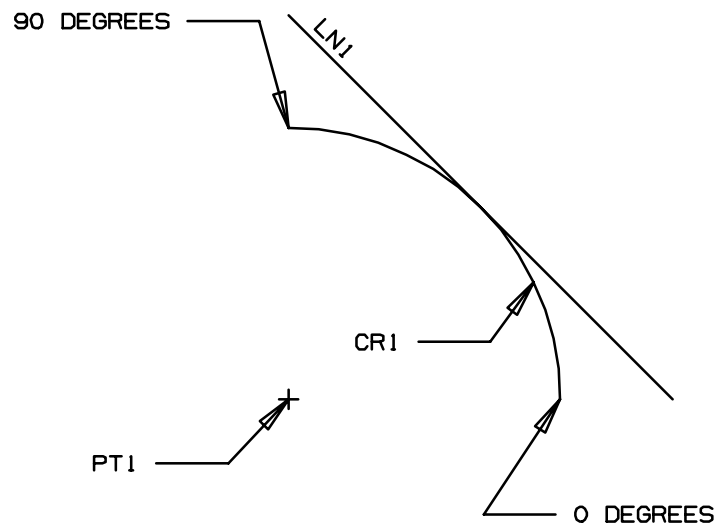


Figure 4–3 Arc with a specified center point and tangent to a line



Circle: Center Point, Point on the Arc

Synopsis

**obj = CIRCLE/CENTER,point1,point2
[,START,start angle,END,end angle]**

Description

Creates a circle by specifying a previously defined point on its arc (point is used to calculate the radius), whose center point is also previously defined. You may also define start and end angles for the circle.

Parameters

CENTER

Minor word that indicates that the center of the circle is an existing point.

point1

A previously defined point which establishes the center of the circle.

point2

A previously defined point through which the arc passes . The point may lie on a non-constructed portion of the arc.

START

The optional parameter START indicates that the starting angle for the arc will be specified.

start angle

The starting angle of the arc as measured from the positive X axis of the work coordinate system.

END

The optional parameter END indicates that the ending angle for the arc will be specified.

end angle

The ending angle of the arc as measured from the positive X axis of the work coordinate system.



Example The example shows the use of the **Center Point, Point on the Arc** statement to create full and partial circles.

Declarations ENTITY/P(4),CR(2)

Geometry Definition P(1)=POINT/1,0
P(2)=POINT/1,.5
P(3)=POINT/3,0
P(4)=POINT/3,.5

Circle Definition CR(1)=CIRCLE/CENTER,P(1),P(2),START,45,END,360
CR(2)=CIRCLE/CENTER,P(3),P(4)

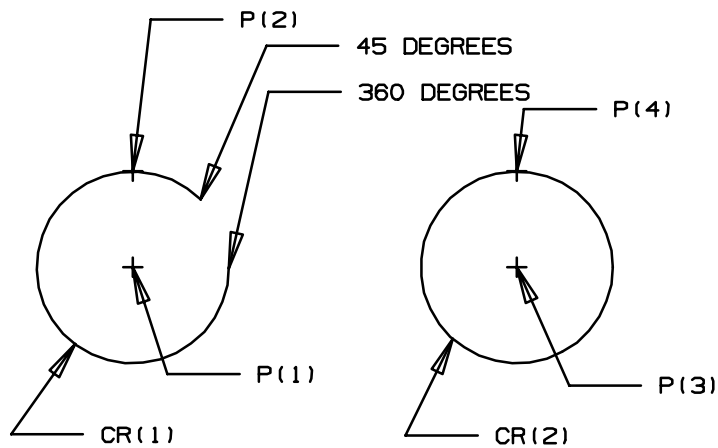


Figure 4-4 Arcs with specified center points and points on the arcs

Circle: Through Three Points

Synopsis

obj = CIRCLE/point1,point2,point3

Description

Creates a circle thru three previously defined points. The points are used to calculate the plane of the arc.

Parameters

point1,point2,point3

Three previously defined points through which the arc passes. The arc is constructed in a counterclockwise direction (with respect to the ZC axis of the WCS) from point1 (start) to point 3 (end). The second point, point 2, may lie on either the constructed or non-constructed portion of the arc. If the three points are collinear, an error message will be displayed.

Example

This example shows the creation of a circle thru three points.

Declarations

ENTITY/P(3),CR1

Geometry Definition

P(1)=POINT/1,0
P(2)=POINT/0,1
P(3)=POINT/0,-1

Circle Definition

CR1 = CIRCLE/P(1),P(2),P(3)

NOTE

The following statement shows the use of the point array to specify the three points in the arc.

CR1 = CIRCLE/P

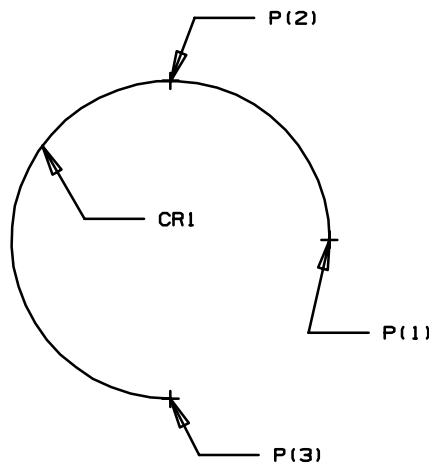


Figure 4-5 An arc through three points



Fillet

The **FILLET** statement creates an arc of a specified radius between two objects.

Filletts are created counterclockwise from the positive X direction of the work coordinate system.

The **FILLET** statement will trim the objects being filleted to the endpoints of the fillet. Trim can be suppressed by specifying the optional minor word NOTRIM.

Table 4–2 Fillet Command Statements

Function	Format
Two Entities, Center Point	FILLET/ <u>obj1</u> , <u>obj2</u> ,CENTER, <u>point</u> , RADIUS,r[,NOTRIM] [,IFERR,label:]
Two Lines, Positional Modifiers	FILLET/“PMOD3”, <u>line1</u> ,“PMOD3”, <u>line2</u> ,RADIUS,r[,NOTRIM] [,IFERR,label:]



Fillet: Two Objects, Center Point

Synopsis

**obj = FILLET/obj1,obj2,CENTER,point,RADIUS,r
[,NOTRIM][,IFERR,label:]**

Description

Creates a fillet between two objects by specifying the objects (e.g. two lines), a point which lies in the same area as the center of the fillet radius, and the radius value.

The center point should be in the general vicinity of the fillet radius center. If two perpendicular lines are being filleted, the point can be anywhere in the quadrant which contains the desired fillet radius. However, if you want to fillet other objects, defining the proper point can be more difficult.

Parameters

obj1,obj2

The two existing objects to be filleted. The objects do not have to intersect, but the maximum distance between the two points of tangency must be less than or equal to twice the specified radius of the fillet. The fillet will be constructed counterclockwise from the first object to the second object.

CENTER

Minor word that indicates that a point will be supplied which will approximate the location of the fillet center.

point

A previously defined point which approximates the location of the fillet center. If the indicated center is invalid, the error message NO FILLET will be displayed. The approximate point will be projected onto the fillet plane along the plane's Z axis.

RADIUS, r

A positive value which establishes the fillet radius. If the radius is invalid, the error message NO FILLET will be displayed.

NOTRIM

If the NOTRIM parameter is included the objects will remain unchanged, if omitted the objects will be trimmed to the points of tangency.

IFERR,label:

Specifies a label to which program execution jumps if an error occurs.

Example This example demonstrates the creation of a fillet between two previously defined lines. NOTRIM suppresses trimming the lines. A previously defined point (PT1) is used to specify the fillet radius center.

Without the point (PT1), this fillet statement could generate four different fillets, one in each quadrant formed as the lines cross. The point (PT1) clearly lies in the lower quadrant which is where the fillet is created.

Declarations ENTITY/PT1, LN1, LN2, FLT1

Geometry Definition PT1 =POINT/0,0
LN1 =LINE/0,1.5,1,-.5
LN2 =LINE/1,1.5,-1,-.5

Fillet Definition FLT1=FILLET/LN1, LN2, CENTER, PT1, RADIUS, .5, NOTRIM

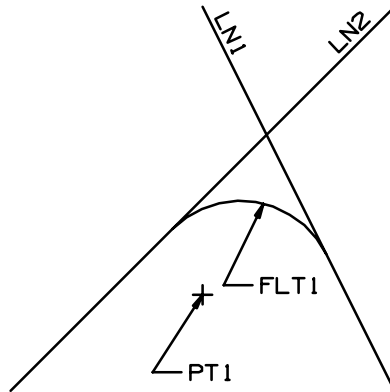


Figure 4-6 Two object fillet without trimming



Example

This example demonstrates the creation of a fillet between two previously defined lines. A previously defined point (PT1) is used to specify the fillet radius center.

Without the point (PT1), this fillet statement could generate four different fillets, one in each quadrant formed as the lines cross. The point (PT1) clearly lies in the lower quadrant which is where the fillet is created.

Declarations

ENTITY/PT1, LN1, LN2, FLT1

Geometry Definition

PT1 = POINT/0,0
 LN1 = LINE/0,1.5,1,-.5
 LN2 = LINE/1,1.5,-1,-.5

Fillet Definition

FLT1 = FILLET/LN1, LN2, CENTER, PT1, RADIUS, .5

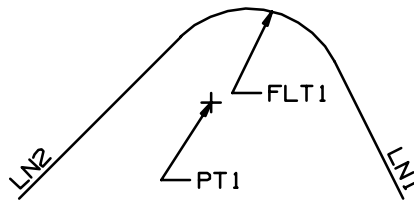


Figure 4–7 Two object fillet with trimming



Fillet: Two Lines, Positional Modifiers

Synopsis

**obj = FILLET/“PMOD3”,line1,”PMOD3”,line2,
RADIUS,r[,NOTRIM] [,IFERR,label:]**

Description

Creates a fillet between two previously defined lines. The fillet radius center is located by specifying its position relative to the objects using positional modifiers. The fillet will be created counterclockwise from the first to the second object.

Parameters

PMOD3

A positional modifier which is used to indicate on which side of the existing line the fillet lies . The direction will be in reference to the work coordinate system.

line1,line2

The two existing lines to be filleted. The lines do not have to intersect, but must be co-planar and the maximum distance between the two points of tangency must be less than or equal to twice the specified radius of the fillet. The fillet will be constructed counterclockwise from the first line to the second line.

RADIUS

Minor word that indicates that the radius of the fillet will be specified.

r

A positive value which establishes the fillet radius. If the radius is invalid, the error message NO FILLET will be displayed.

NOTRIM

If the **NOTRIM** parameter is included the objects will remain unchanged, if omitted the objects will be trimmed to the points of tangency.

IFERR,label:

Specifies a label to which program execution jumps if an error occurs.



<i>Example</i>	This example demonstrates the creation of a fillet between two lines where the center of the fillet is defined using positional modifiers.
<i>Declarations</i>	ENTITY/LN1, LN2, FLT1
<i>Geometry Definition</i>	LN1 =LINE/-1,0,1,0 LN2 =LINE/0,-1,0,1
<i>Fillet Definition</i>	FLT1 =FILLET/YLARGE, LN1, XSMALL, LN2, RADIUS, .5, \$ NOTRIM

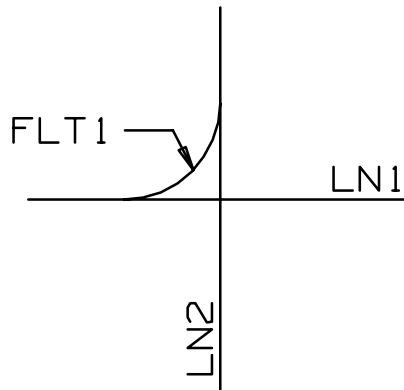


Figure 4–8 Lines filleted in the second quadrant



Example This example demonstrates the creation of a fillet between two lines by specifying the fillet center using positional modifiers.

Declarations ENTITY/LN1, LN2, FLT1

Geometry Definition LN1 =LINE/-1,0,1,0
LN2 =LINE/0,-1,0,1

Fillet Definition FLT1 =FILLET/YSMALL, LN1, XLARGE, LN2, RADIUS,.5, \$
NOTRIM

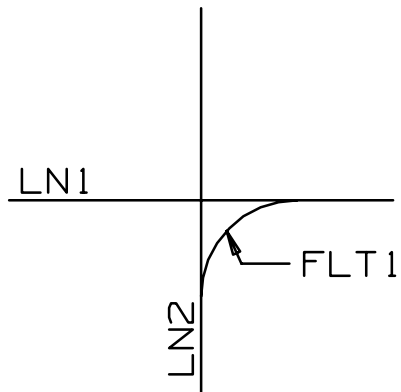


Figure 4–9 Lines filleted in the fourth quadrant



Circle EDA's

There are several EDA's which access the geometric properties of arcs. These EDA's work on objects that were created using either the CIRCLE/ or the FILLET/ command. Some of the EDA's are READ/WRITE, which means that the properties can be accessed and edited. Using EDA's in the edit mode can be a convenient method for making extensive changes. For example, you could write a program that would READ the radius of every arc in a part, and edit any .250 radii to .253, if press fits for that size hole were all replaced with slip fits.

The X, Y, and Z axis matrix values represent axes of the coordinate system of the arc. The arc lies in the X–Y plane, with the arc angle measured counterclockwise from the positive X axis.

Geometric Property	EDA Symbol	Access Type	Data Type
Center (X,Y,Z)	&CENTER(<u>obj</u>)	RW	N(3)
Radius	&RADIUS(<u>obj</u>)	RW	Number
Start angle	&SANG(<u>obj</u>)	RW	Number
End angle	&EANG(<u>obj</u>)	RW	Number
Start point (X,Y,Z)	&SPOINT(<u>obj</u>)	RW	N(3)
End point (X,Y,Z)	&EPOINT(<u>obj</u>)	RW	N(3)
Arc length	&LENGTH(<u>obj</u>)	RO	Number
X axis matrix values	&XAXIS(<u>obj</u>)	RO	N(3)
Y axis matrix values	&YAXIS(<u>obj</u>)	RO	N(3)
Z axis matrix values	&ZAXIS(<u>obj</u>)	RO	N(3)



Example

This example demonstrates the use of the circle EDA's to edit the start and end angles of an arc.

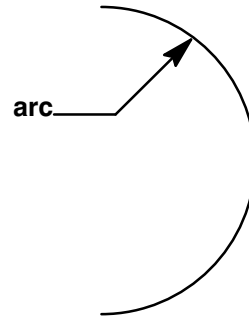


Figure 4–10 Original arc, starting at 270 degrees, ending at 90 degrees

```
ENTITY/ arc  
arc = CIRCLE/ 0, 0, 1, START, 270, END, 90  
&SANG(arc) = 0  
HALT
```

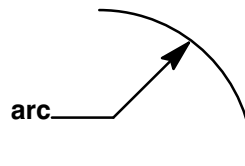


Figure 4–11 Edited arc, starting at 0 degrees, ending at 90 degrees



Entity Independent EDA's

There are several EDA's that allow information about objects to be extracted and modified.

Geometric Property	EDA Symbol	Access Type	Data Type
Font	&FONT(obj)	RW	Number [1..7]

For the &FONT EDA, the following values are expected:

- 1 = &SOLID
- 2 = &DASHED
- 3 = &PHANTM
- 4 = &CLINE
- 5 = Dotted
- 6 = Long Dashed
- 7 = Dotted Dashed

Example

```
ENTITY/ arc
arc = CIRCLE/0,0,1, START,270, END,90
&FONT(arc) = &DASHED
```

Geometric Property	EDA Symbol	Access Type	Data Type
Color	&COLOR(obj)	RW	Number [1..15]

For the &COLOR EDA, the following values are expected:

- 1 = &BLUE
- 2 = &GREEN
- 3 = &CYAN
- 4 = &RED
- 5 = &MAGENT
- 6 = &YELLOW
- 7 = &WHITE
- 8 = &OLIVE
- 9 = &PINK
- 10 = &BROWN
- 11 = &ORANGE
- 12 = &PURPLE
- 13 = &DKRED
- 14 = &AQUAMR
- 15 = &GRAY

Example

```
ENTITY/ arc
arc = CIRCLE/ 0, 0, 1, START, 270, END, 90
&COLOR(arc) = &MAGENT
```



Geometric Property	EDA Symbol	Access Type	Data Type
Line Width	&LWIDTH(obj)	RW	Number [1..3]

For the **&LWIDTH** EDA, the following values are expected:

- 1 = Normal**
- 2 = Thick (heavy)**
- 3 = Thin**

Example

ENTITY/ arc

```
arc = CIRCLE/ 0, 0, 1, START, 270, END, 90
&LWIDTH(arc) = 2
```

Geometric Property	EDA Symbol	Access Type	Data Type
Layer	&LAYER(obj)	RW	Number [1..256]

Example

ENTITY/ arc1, arc2

```
arc2 = CIRCLE/ 0, 0, 1, START, 270, END, 90
&LAYER(arc2) = &LAYER(arc1)
```



Nesting

You may nest definitions of objects within other definitions.

- The nested statement must have parentheses around it.
- A maximum of 10 objects may be nested with each other.

Nesting can be useful, but you can also carry it to extremes. Nesting will make a program difficult to read, to debug, and to revise, and may have a great effect on the run time.

Example

```
ENTITY/P1 , P2 , LN1
P1=POINT/0 , 1.5
P2=POINT/2.2 , 3.7
LN1=LINE/P1 , P2
HALT
$$ PREVIOUS PROGRAM COULD HAVE BEEN WRITTEN
```

```
ENTITY/P1 , P2 , LN1
LN1=LINE/ ( P1=POINT/0 , 1.5 ) , ( P2=POINT/2.2 , 3.7 )
HALT
```

Example

```
$$ CREATE POINT AT INTERSECTION OF 2 LINES
ENTITY/P1 , P2 , P3 , P4 , P5 , LN1 , LN2
P5=POINT/INTOF , ( LN1=LINE/$
    ( P1=POINT/1 , 1 ) , ( P2=POINT/3 , 1.5 ) ) , ( LN2=LINE/$
    ( P3=POINT/1.5 , 6.2 ) , ( P4=POINT/3.2 , 2.7 ) )
HALT
```

Activity: Sheet Metal Part

Write a program to draw the sheet metal part pictured on the following page. Follow the steps below.

- Use any geometry definition necessary.
- The dimensions of the part are 5 X 6, with filleted corners, containing two holes and a pocket with rounded ends.

The radii of the circles and arcs are one-sixth the distance between the center of the part and the corners of the part (before filleting). To get this distance, use the formula:

$$\text{rad}=\text{SQRTF}((\text{hgt}/2)**2+(\text{wid}/2)**2)/6$$

- Construct the geometry using the center of the part as your starting point. All objects should be created in relation to the center of the part. Do not start at one of the corners.

When you finish the assignment, demonstrate the program to your instructor.

Optional Tasks:

- Make the color of the part edge cyan, with yellow hole and slot features.
- Make the font of the part solid, with dashed hole and slot features.
- Make the line width of the part edge normal, with thin hole and slot features.

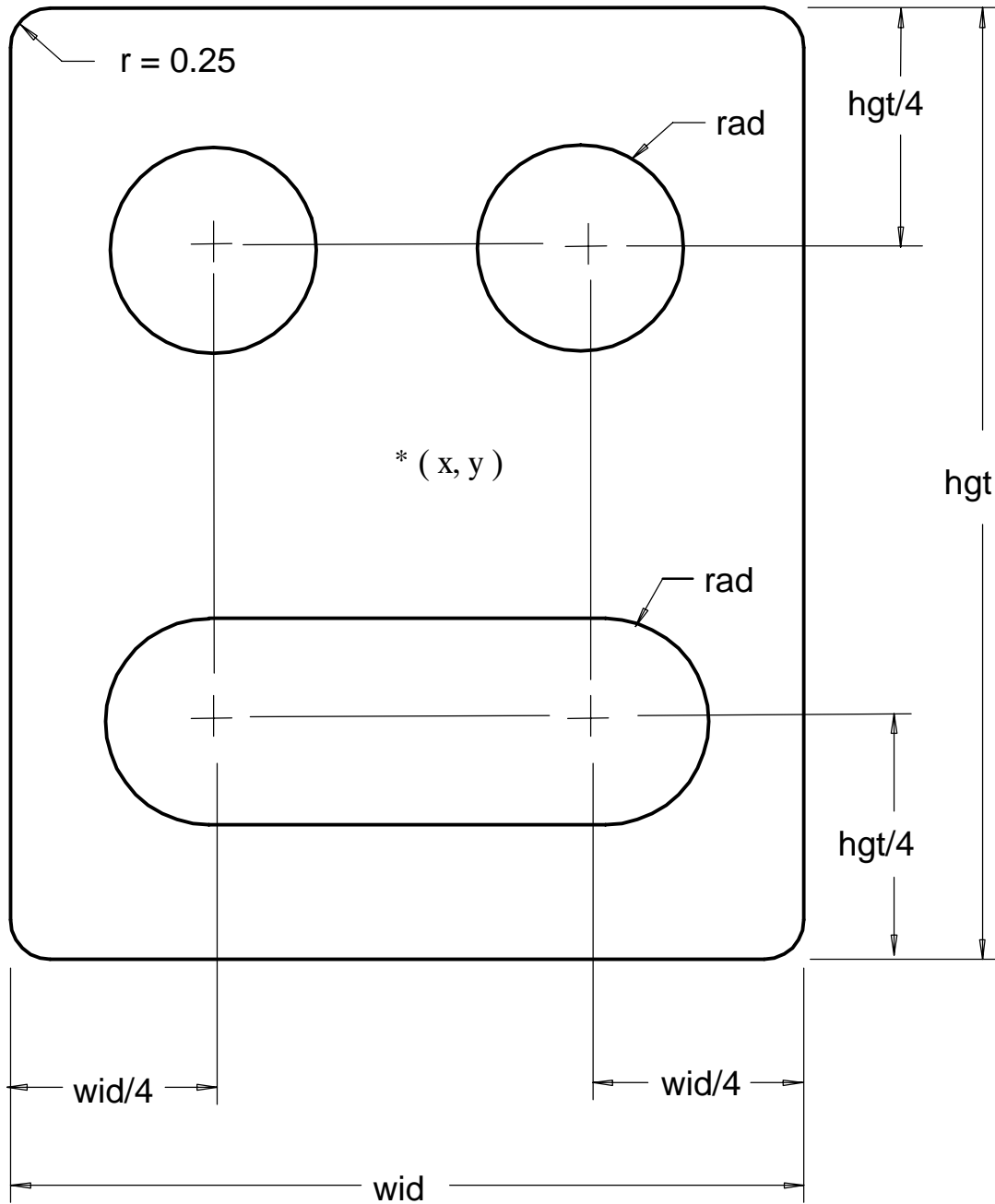


Figure 4–12 Sheet Metal Part

(This Page Intentionally Left Blank)



Controlling Program Execution

Lesson 5

Objectives

- Use the commands for branching, logic, and looping.
- Build data checking into the user interface.
- Write error handling code.
- Write structured logic.



Introduction

Up to this point, we have discussed GRIP programs that execute sequentially from beginning to end. However, we typically want the computer to execute different portions of code depending on various conditions. These conditions may depend on information provided by the user or other stored values. The commands for branching allow altering the flow of the program. When overused, they can make programs difficult to maintain and understand.

Table 5–1 Branching, loop and logic Statements

Function	Format
Branch always	JUMP/label1:
Branch using the integer value of the expression.	JUMP/ [label:]+, expression
Loop to label:. Iterate from 'begin' to 'end'	DO/label:,index,\$ begin, end[,increment]
Logical IF	IF/logical expression, \$ GRIP statement
Block IF	IFTHEN/logical expression GRIP statements as required [ELSEIF/logical expression GRIP statements as required] [ELSE GRIP statements as required] ENDIF



Suggested Labeling Guidelines

Most commands in Table 5–1 use labels. Labels can be thought of as tags that identify special locations in your code. A label is a name which may precede any GRIP statement (except the PROC/ statement). In GRIP, the following rules apply for labels:

- Labels can contain up to 32 characters.
- The first six characters of each label must be unique.
- All labels must start with an alphabetic character, with the remaining characters alphanumeric.
- There can be no blank characters.
- A label must end with a colon (:) character.

NOTE GRIP allows a total of 1000 labels per subroutine. A DO loop generates two labels, a CALL command generates one label, and the IFTHEN/ELSE/ELSEIF/ENDIF uses two labels plus one for every ELSEIF. These labels count against the total you can use.

The following section describes a recommended labeling standard. These are not rules and should not be interpreted as such.

1. When reading a GRIP program, labels should “stand out”. It is suggested that all labels start in column one, with remaining statements starting after column eight.

2. Each label should describe its purpose by using a mnemonic to describe the purpose of the label. Mnemonics include:

- BCKxxx:** *Precedes an interactive command, providing a position for execution to return when the user selects the Back option on the dialog box.*
- DOLxxx:** *Used to construct “DO” loops.*
- WHLxxx:** *Used to simulate “WHILE” loops, which are common constructs available in other high level programming languages*
- LBLxxx:** *A generic label to be used sparingly for a variety of purposes including non–fatal error handling.*
- ERRxxx:** *Indicates code to be executed when an error condition occurs. May precede a “MESSG” statement.*
- FAIL:** *A position in a program or subroutine for use in error handling when a fatal error is encountered.*
- RTN:** *Indicates the return point in a subroutine. Precedes the “RETURN” statement.*
- CANCEL:** *Marks the end of a GRIP program. Precedes the “HALT” statement.*

3. Use a sequential series of numbers as a suffix to the previously defined mnemonics. This makes it easier for the reader to locate labels when many labels are used. Alternate the numbers in even units of 10 when writing the code, in order to leave room for inserting more labels. For example:

BCK010:

BCK015:

WHL020:

LBL025:

DOL030:

ERR040:

The labels BCK015: and LBL025: can assume to have been added after the code was originally written.

The rules for generating labels are not restrictive, allowing for a wide variety of labeling styles. When programmers use different methods for naming labels, understanding GRIP programs becomes difficult. You should follow a standard labeling convention which can be easily understood by other programmers.

JUMP: Unconditional Branching

Synopsis

JUMP/label:

Description

This statement allows you to perform unconditional branching, which causes the program to branch to the statement containing the specified label. This feature is generally used to bypass sections of the program which are controlled by conditional branching statements or as the executable part of a logical **IF** statement.

Parameters

label:

Label to which program execution will jump.

Example

This example demonstrates the use of the **JUMP** statement in an **IF** statement. A **JUMP** statement is also used to skip over a portion of the program which is only executed under certain conditions.

```
IF/mtype= =1, JUMP/ALUM:
IF/mtype= =2, JUMP/STL:
IF/mtype= =3, JUMP/OTHR:
```

```
ALUM:
```

```
dens = .100
JUMP/LBL030:
```

```
STL:
```

```
dens = .289
JUMP/LBL030:
```

```
OTHR:
```

```
PRINT/'Density not defined'
```

```
LBL030:
```

NOTE

In the example above, if the variable *mtype* is equal to 1, the variable *dens* is assigned a value of .100 and the statements between the labels *STL:* and *LBL030:* are bypassed.

JUMP: Conditional Branching

Synopsis

JUMP/label: +, [expression]

Description

Conditional branching is the computed or conditional **JUMP** statement that causes the program to branch to one of several locations based on the integer portion of a supporting expression or variable. The integer may be set in many ways, either by a numerical expression (ntype = a+b) or by a GRIP interactive statement such as **PARAM**, **CHOOSE**, etc.. If the number being considered is not an integer (4.25), the integer portion of the expression or variable, which is also referred to as the index, is obtained by truncating without rounding in either direction (4.25 would be evaluated as 4). The resulting integer will cause the program to branch to the label specified by that location in the label list which may consist of up to 43 labels. If a label does not exist in the indicated location (a field between two commas L1:,L2:,,L4:) the program will continue with the next statement in the program. This statement is ignored if the resulting integer value is a negative, is zero, or is greater than the number of labels provided in the statement.

Parameters

label: +

A desired number of unique label names to which the program will branch. A label must end with a colon (:).

expression

An expression results in a numeric value. The numeric value may be set by the expression or by a previous GRIP statement.

Example

This example demonstrates the use of a **JUMP** statement to branch based on the results of expressions in the statement.

```
JUMP/ALUM:, STL:,OTHR:, mtype
ALUM:
    dens = .100
    JUMP/LBL030:
STL:
    dens = .289
    JUMP/LBL030
OTHR:
    PRINT/'Density not defined'

LBL030:
```

The three IF statements from the previous example are replaced by a single conditional JUMP statement yielding the same result.



DO: Program Loop

Synopsis

DO/label:index variable,start,end[,increment]

Description

The following provides an effective means of performing a given operation a specified number of times. The operation commonly referred to as a **DO** loop is a segment of the program which starts with the statement containing the word **DO**, followed by a slash and a label, and ends with the statement or blank statement which starts with the specified label.

Additional **DO** loops, referred to as nested loops, may exist inside of a loop, referencing either the same label or a separate label. If a separate label is used in a nested loop, both the **DO** statement and the specified label statement must exist within the outer loop.

NOTE

GRIP allows a total of 1000 labels per subroutine. A **DO** loop generates two labels. These labels count against the total you can use.

Acceptable Jump Situations:

- A jump may be made from a statement inside of a loop to a label which is also inside of the loop.
- A jump may be made from a statement inside of a loop to a label which is outside of the loop (this will terminate the loop).

Unacceptable Jump Situation:

- A jump may not be made from a statement outside of a loop to a label which is inside of a loop.

Parameters

label:

Specifies either the last executable statement in the loop or a blank statement following the last executable statement in the loop.



index variable

A variable which registers the current value of the loop. When the loop is entered, the index variable will be set to the start value; when the loop is finished, the index variable will be equal to the end value plus the increment value.

start

The starting value for the loop which may be either a constant, an expression, or a variable.

end

The ending value for the loop which may be either a constant, an expression, or a variable.

increment

The optional increment value for the loop which may be either a constant, an expression, or a variable. If not specified the increment default is 1.

Example This example demonstrates the creation of several points using a nested **DO** loop.

Declarations ENTITY/pt(2,4)
NUMBER/delt(2),x(4),y(4),i,j

DATA/delt,-2,2
DATA/x,-.5,0,.5,0
DATA/y,0,-.5,0,.5

Do Statements DO/ENDO2:,i,1,2
 DO/ENDO1:,j,1,4
Geometry Definition pt(i,j)=POINT/delt(i)+x(j), y(j)
 ENDO1:
 ENDO2:



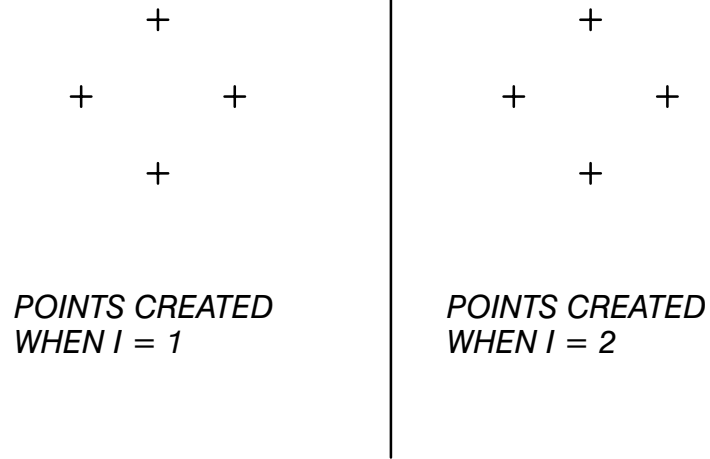


Figure 5–1 Nested DO loops

IF: Logical IF

Synopsis

IF/logical expression,statement

Description

This statement provides for the execution or exclusion of a statement based on the results of a logical expression. If the results of the logical comparison is true, for example: does 1 equal 1, IF/1==1, the following statement will be executed. If the results of the logical comparison is false, for example: does 1 equal 2, IF/1==2, the following statement will not be executed.

The order of evaluation for the arithmetic and logical operators is important in obtaining the desired results.

Order	Operator
First	Arithmetic Operators: + Addition - Subtraction * Multiplication / Division ** Exponentiation
Second	Logical Operators: == Equal to <> Not equal to < Less than <= Less than or equal to > Greater than >= Greater than or equal to
Third	Boolean Operator: NOT Complement
Fourth	Boolean Operators: AND Both OR Either



*Parameters***logical expression**

A logical expression which consists of two values separated by a logical operator. The expression may consist of any combination of constants and numerical variables or any combination of string literals and string variables. The following table is a list of logical operators which perform the comparisons (the symbols < and > represent the less than and greater than symbols respectively).

Operator	Comparison
==	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

NOTE EDA's are not allowed in logical expressions.

statement

Any executable GRIP statement such as:

JUMP/label:

A=1

DELETE/LN1

Exceptions are: **DO** and **DECLARATION** statements.

Example

This example demonstrates the use of the logical **IF** statement. Logical operations may be performed on numerical variables and/or constants.

```
scoord=&SPOINT(ln1)
ecoord=&EPOINT(ln1)
```

```
IF/scoord(2)==ecoord(2), DELETE/ln1
```

In the example above, the line ln1 will be deleted if it is horizontal.

Example The following example shows how logical operations may be performed on an entity variable.

Declarations ENTITY/PT1, NT1

```
IF/PT1<>&NULENT,JUMP/LBL020:  
    PT1=POINT/2,2  
LBL020:  
    NT1=NOTE/DELTA,PT1,.25,.25,'BILL OF  
MATERIALS'  
LBL040:
```

To avoid an error in the **NOTE** statement, the GPA **&NULENT** is used in the **IF** statement to check if point PT1 exists. If it does, execution will jump to label LBL020: and the note will be created. Otherwise, PT1 will be created then the note will be created.

Example This example demonstrates that logical operations may be performed on string variables and/or string literals.

```
IF/partno<>&NULSTR, nt=NOTE/x,y,partno
```

In the example above, the GPA **&NULSTR** was used to check if the string partno had a value to avoid an error in the **NOTE** statement.

Example The previous examples have shown logical operations on simple variables, constants, and string literals. Logical operations, however, may also be performed on arrays or subranges of arrays which are of similar type and size.

Declarations NUMBER/A(3),B(3)
DATA/A,1,2,3
DATA/B,1,4,3

If Statement IF/A==B, JUMP/LBL010:
MESSG/'THE ARRAYS ARE NOT EQUAL'
LBL010:

Since A in the example above, which was set to 1, 2 and 3, is not equal to B, which was set to 1, 4 and 3, the message THE ARRAYS ARE NOT EQUAL will be displayed.

The logical array comparison in the program above would be equal to the individual comparison in the program below.

Declarations NUMBER/A(3),B(3)
DATA/A,1,2,3
DATA/B,1,4,3

If Statement DO/DOL010:,I,1,3
IF/A(I)<>B(I), JUMP/LBL020:
DOL010:

JUMP/LBL030:

LBL020:
MESSG/'THE ARRAYS ARE NOT EQUAL'
LBL030:

BLOCKIF: Block If

Synopsis

```
IFTHEN/e1  
    block1  
[ELSEIF/e2  
    block2]  
    .  
    .  
    .  
[ELSE  
    blockn]  
ENDIF
```

Description

The **BLOCKIF** statements allow you to conditionally execute blocks (or groups) of GRIP statements. The four **BLOCKIF** statements are:

- **IFTHEN**
- **ELSEIF**
- **ELSE**
- **ENDIF**

These statements are used in **BLOCKIF** constructs. A block is a sequence of zero or more complete GRIP statements. Each sequence is called a statement block.

Each **BLOCKIF** statement, except the **ENDIF** statement, has an associated statement block. The statement block consists of all the statements following the **BLOCKIF** statement up to (but not including) the next **BLOCKIF** statement in the **BLOCKIF** construct. The statement block is conditionally executed based on the value(s) of the logical expression(s) in the preceding **BLOCKIF** statements.

The **IFTHEN** statement begins a **BLOCKIF** construct. The block following it is executed if the value of the logical expression in the **IFTHEN** statement is true.

The **ELSEIF** statement is an optional statement that specifies a statement block to be executed if no preceding statement block in the **BLOCKIF** construct has been executed, and if the value of the logical expression in the **ELSEIF** statement is true.



The **ELSE** statement specifies a statement block to be executed if no preceding statement block in the **BLOCKIF** construct has been executed. The **ELSE** statement is optional. However, if the **ELSE** statement is present, its statement block must be immediately followed by the **ENDIF** statement.

The **ENDIF** statement terminates the **BLOCKIF** construct.

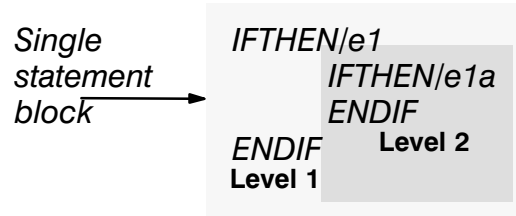
NOTE GRIP allows a total of 1000 labels per subroutine. The **IFTHEN/ELSE/ELSEIF/ENDIF** uses two labels plus one for every **ELSEIF**. These labels count against the total you can use.

Parameters

IFTHEN

The **BLOCKIF** statement which begins the construct. It evaluates the logical expression passed to it. If the expression is *true*, it executes the accompanying block. Up to 32 levels of **IFTHEN** statements can be nested per block as shown in Figure 5–2.

Nest up to 32 levels per block



<u>Levels</u>	
1	<i>IFTHEN</i>
2	<i>IFTHEN</i>
2	<i>ENDIF</i>
1	<i>ELSEIF</i>
2	<i>IFTHEN</i>
3	<i>IFTHEN</i>
3	<i>ENDIF</i>
2	<i>ENDIF</i>
1	<i>ENDIF</i>

Figure 5–2 Levels of IFTHEN statements per block

e1

The logical expression passed to the **IFTHEN** statement.



block1

A set of GRIP statements which are executed if the preceding e1 expression is true.

ELSEIF

An optional **BLOCKIF** statement that specifies a statement block to be executed if no preceding statement block of the **BLOCKIF** construct has been executed, and if the value of the logical expression in the **ELSEIF** statement is true. Each **IFTHEN** construct can contain up to 36 **ELSEIF** statements.

e2

The logical expression passed to the **ELSEIF** statement.

block2

A set of GRIP statements which are executed if the preceding e2 expression is true.

ELSE

An optional **BLOCKIF** statement which is executed if no previous statement block in the **BLOCKIF** construct has been executed.

blockn

A set of GRIP statements which the system executes if no previous statement block in the **BLOCKIF** construct has been executed.

ENDIF

The **BLOCKIF** statement which terminates the construct.

Example

The simplest **BLOCKIF** construct consists of the **IFTHEN** and **ENDIF** statements. This construct conditionally executes one statement block.

```
IFTHEN/a == b
      MESSG/'ARRAY A IS EQUAL TO ARRAY B'
ENDIF
```

The statement first evaluates the logical expression $a == b$. If this value is true, then the program displays the message **ARRAY A IS EQUAL TO ARRAY B**. If the value is false, then the statement block is not executed and control transfers to the next executable statement after the **ENDIF** statement.



The following example contains a **BLOCKIF** construct using the **ELSEIF** statement.

```
IFTHEN/A == B
    MESSG/'Array A is equal to array B'
ELSEIF/A == C
    MESSG/'Array A is equal to array C'
ENDIF
```

If the value of a is equal to b, the program displays the message *Array A is equal to array B*. If the value of A is equal to C, then the program displays the message *Array A is equal to array C*. But if neither expression is true, neither statement block is executed and control transfers to the next executable statement after the **ENDIF** statement.

Example

The following example contains a **BLOCKIF** construct using both the **ELSEIF** and **ELSE** statements.

```
IFTHEN/A > B
    MESSG/'Array A is greater than array B'
ELSEIF/A < B
    MESSG/'Array A is less than array B'
ELSE
    MESSG/'Array A is equal to array B'
ENDIF
```

In this case, if the value of A is greater than B, the program displays the message *Array A is greater than array B*. If the value of a is less than b, the program displays the message *Array A is less than array B*. But if neither case is true (the value of a is equal to b), then the program displays the message *Array A is equal to array B*.

Example

The following example contains a **BLOCKIF** construct using several **ELSEIF** statements with the **ELSE** statement.

```
IFTHEN/a>b
    d=b
    f=a-b $$ End of first block

ELSEIF/a>c
    d=c
    f=a-c $$ End of the second block

ELSEIF/a>z
    d=z
    f=a-z $$End of third block
```



```

ELSE
    d=0
    f=a  $$ End of fourth block
ENDIF
    
```

There are four statement blocks in this example (one for each **BLOCKIF** statement except **ENDIF**). If a is greater than b, the first block is executed. If a is not greater than b but is greater than c, the second block is executed. If a is not greater than b or c but is greater than z, the third block is executed. If a is not greater than b, c, or z, the fourth block is executed.

Example The following example illustrates a nested **BLOCKIF** construct.

```

IFTHEN/A=B
    L=LINE/A,B+1
ELSEIF/A>B
    
```

```

IFTHEN/B > C
    L = LINE/A,C
ELSE
    L = LINE/A,B
ENDIF
    
```

} *Nested BLOCKIF: Note that this block may contain up to 36 ELSEIF statements independent of the outer block.*

```

ELSE
    L=LINE/B,A
ENDIF
    
```



IF Boolean Operators

The operator **AND** functions the same in a logical **IF** and **IFTHEN** statement as the conjunction **AND** does in a sentence. For example, if an operation depended on the results of the two simple expressions $A==B$ and $C==D$, the two logical **IF** statements, **IF/A==B** and **IF/C==D**, with the necessary labels and branching statements could be programmed to accomplish this task. However, these two simple expressions could be combined to form the compound logical **IF** statement, **IF/A==B AND C==D** and thereby reduce the amount of necessary programming.

Example Using the AND boolean

The following code accomplishes a task by using a sequence of logical **IF** statements with simple expressions.

```

IF/A<>B, JUMP/A1:
IF/C==D, PRINT/'BOTH EXPRESSIONS ARE
TRUE'
A1:

```

The same task can be accomplished using a single logical **IF** statement with a boolean operator.

```

IF/A==B AND C==D, $
PRINT/'BOTH EXPRESSIONS ARE TRUE'

```

Like the operator **AND**, the operator **OR** also functions the same in a logical **IF** and **IFTHEN** statement as the conjunction **OR** does in a sentence. For example, if an operation depended on the results of the two simple expressions $A==B$ or $C==D$, the two logical **IF** statements, **IF/A==B** or **IF/C==D**, with the necessary labels and branching statements could be programmed to accomplish this task. However, these two simple expressions could be combined to form the compound logical **IF** statement, **IF/A==B OR C==D** and thereby reduce the amount of necessary programming.

Example Using the OR boolean

The following example accomplishes a task by using a sequence of logical **IF** statements with simple expressions.

```
IF/A == 10, JUMP/LBL010:  
IF/B <> 10, JUMP/LBL020:  
LBL010:  
D = C/(A+B)  
LBL020:
```

The following accomplishes the same task by using a logical **IF** statement with a compound expression.

```
IF/A == 10 OR B == 10, D = C/(A+B)
```

Example Using the NOT boolean

The operator **NOT** takes the result of the previously evaluated logical expression and compliments or reverses it. For example:

```
IF/NOT(A == B), JUMP/LBL100:
```

The benefit of this feature is that both conditions of an expression may be tested without changing the logical operators.



Error Handling

Many statements in the GRIP programming language contain the **IFERR** minor word for trapping error conditions. The **IFERR** word is used with a label. When an error occurs, program execution will be altered, resuming at the statement immediately following the **IFERR** label.

There are several ways to write error handling code depending on whether the error is considered a fatal error or a recoverable error, the complexity of the program, etc. This section will therefore explain a variety of techniques that can be used to handle errors with GRIP commands.

Non–Fatal Error Recovery

Often errors are encountered that may be due to incorrect information selected or entered by the user. When these problems occur, your program should explain what happened and provide the opportunity to correct the problem. The coding example below illustrates this technique.

Example IFERR for non fatal errors

```

ENTITY/ crv(2), ref_pt, int_pt
NUMBER/ resp, xc, yc, zc
BCK010:
MASK/ 3, 5, 6, 7, 8, 9
IDENT/ 'Select TWO Curves for Intersection',$
      CURSOR, xc, yc, zc, crv, resp
JUMP/ BCK010:, CANCEL:, resp
ref_pt = POINT/ xc, yc, zc
int_pt = POINT/ ref_pt, INTOF, crv(1), crv(2),$
      IFERR, LBL020:

      JUMP/ LBL030:  $$ No error finding INTOF, proceed.
LBL020:           $$ Could not intersect curves.
      DELETE/ ref_pt
      MESSG/ 'Could not create intersection between curves',$
          'Please try again.'
      JUMP/ BCK010:
LBL030:
$$ Normal processing.

      DELETE/ ref_pt
      .
      .
CANCEL:
      HALT

```

Fatal Error Recovery, Method 1

Often, when an error is encountered, it is not possible to continue the program. In this case, you should make the person running the program aware of the problem and halt the program. The example below shows a simple way of doing this.

Example IFERR for fatal errors

```
ENTITY/ crv(2), ref_pt, int_pt
.
.
.
$$
$$ A section of code creates crv(1..2) and ref_pt
$$
    pt = POINT/ INTOF, crv(1), crv(2), IFERR, ERR100:
$$
$$ Now fillet the two curves.
$$
    fil = FILLET/ crv(1), crv(2), CENTER, ref_pt, $
           RADIUS, .25, IFERR, ERR110:
$$
$$ Remainder of normal processing
$$
.
.
.
JUMP/ CANCEL:

ERR100:
    MESSG/ 'Could not intersect curves'
    JUMP/ CANCEL:
ERR110:
    MESSG/ 'Could not create fillets'
    DELETE/pt

CANCEL:
    HALT
```

The error handling technique described above is efficient for short programs. However, for larger applications, many error labels and **MESSG** statements might prove cumbersome. The method on the next page addresses this problem.



Fatal Error Recovery, Method 2

In this example, the multiple message statements of the previous example have been removed through the use of a string array (ERRMSG) which stores error messages. Before each command which might cause an error, the index (errno) is set to a value corresponding to a position in the error message array. The **FAIL:** label is a common point where control will be passed by all statements using the **IFERR** word when an error occurs.

Example IFERR for fatal errors; Text array of messages

```

ENTITY/ crv(2), ref_pt, int_pt
NUMBER/ errno
STRING/ ERRMSG(10, 60)

DATA/ ERRMSG,    $ $$ Error messages for fatal errors
    'Could not intersect curves',    $
    'Could not create fillets',      $
    'Could not merge template part', $
    'Could not file part'
.
.
.
$$
$$ A section of code creates crv(1..2) and ref_pt
$$
    errno = 1
    pt = POINT/ INTOF, crv(1), crv(2), IFERR, FAIL:
$$
$$ Now fillet the two curves.
$$
    errno = 2
    fil = FILLET/ crv(1), crv(2), CENTER, ref_pt, $
        RADIUS, .25, IFERR, FAIL:
$$
$$ Remainder of normal processing
$$
.
.
.
JUMP/ CANCEL:
FAIL:
    MESSG/ ERRMSG(errno)
CANCEL:
    HALT

```

(This Page Intentionally Left Blank)



Basic Interactive Commands

Lesson 6



Objectives

- Demonstrate an understanding of the types of interactive commands available.
- Use interactive commands in conjunction with the branching and logic commands to build a user interface.

Interactive Commands



A variety of interactive commands are available in the programming language. These commands display menus on the dialog box and graphics screen for the user to enter information. Many of these commands provided the Back and Cancel options so that the user can reverse a previously made decision or exit the GRIP program.

Table 6–1 Interactive commands

Write to Dialog Box	MESSG/[TEMP,]string1[,string2]
Screen position pick	POS/ 'message', x, y, z, resp
Screen position menu	GPOS/ 'message', x, y, z, resp
Obtain real and/or integer information	PARAM/ 'message', {'option',[INT], variable} [, ALFACT, 'message',] resp

<i>Name</i>	MESSG Write to Dialog Box
<i>Synopsis</i>	MESSG/[TEMP,]string1[,string2]
<i>Description</i>	<p>This statement displays messages in a dialog box or on the status line. It can be used to tell the user during program execution that certain tasks have been accomplished, or that the program is working on a statement which may take some time to execute.</p> <p>CAUTION We do not recommend using the asterisk character (*) in menu prompts or menu options.</p> <p>NOTE Strings may automatically convert to a string where the first character is uppercase followed by lowercase characters if the string exists in the Native Language Menu (NLM) database.</p> <p>TEMP Minor word that indicates that program execution is to continue and that the specified message is to be displayed on the status line. If omitted, program execution is halted until OK is chosen.</p> <p>string1,string2 Strings which may consist of up to 60 characters each. These strings are displayed in a dialog box with the optional second parameter under the first. Only two strings are allowed per message.</p>
<i>Example</i>	This example demonstrates the creation of a sculptured surface. The MESSG statement is used to tell the user during program execution, that the surface is being calculated.
<i>Declarations</i>	ENTITY/L(3),P(4),SPL,SCULPT
<i>Geometry Definition</i>	L(1)=LINE/0,0,0,4,2,1 L(2)=LINE/0,0,0,0,4,1 L(3)=LINE/0,4,1,5,4,2 P(1)=POINT/ENDOF,XLARGE,L(1) P(2)=POINT/4,2,.5 P(3)=POINT/4.25,3,1 P(4)=POINT/ENDOF,XLARGE,L(3) SPL=SPLINE/P DELETE/P

First Message MESSG/TEMP,'CALCULATING SURFACE'
SCULPT=SSURF/PRIMA,L(1),L(3),CROSS,L(2),SPL

Second Message MESSG/TEMP,'SURFACE CREATED'

Third Message MESSG/'SEL EC TO CONTINUE'

The first **MESSG** statement displays the message until the surface is created and the second **MESSG** statement is executed. Since the **TEMP** minor word is used, the program proceeds automatically. The third **MESSG** statement does not contain the **TEMP** minor word, therefore the program pauses until OK is chosen.

Typical Interactive Menu Flow

The following outline provides a methodology for adding interactive commands to a GRIP program.



- Back Label

BCK010:

- Interactive Command

POS/
GPOS/
CHOOSE/
MCHOOS/
PARAM/
IDENT/
TEXT/

- Interpret the Interactive Response

JUMP/

or

IFTHEN/
ELSEIF/
ELSE/
ENDIF

- Act on the Response or Parameters Entered Interactively
- Cancel Label

CANCEL:
HALT



The last field in each interactive command is a 'response'. This is a numeric variable which is assigned a value based on the user's action. Each command assigns different numeric variables to the response field. Most of the interactive commands should be followed with a JUMP or block IF statement to control program branching.

Example Interactive command using Conditional Branching

```

BCK010:
    CHOOSE/'Select Principle Axis', $
        'X AXIS', $
        'Y AXIS', $
        'Z AXIS', resp
    JUMP/ BCK010:, CANCEL: ,,, $
        LBL020:, LBL030:, LBL040:, resp

LBL020: (Option 1)
:
.
LBL030: (Option 2)
:
.
LBL040: (Option 3)

```

Example Interactive command and a Block IF

```

BCK010:
    CHOOSE/'Select Principle Axis', $
        'X AXIS', $
        'Y AXIS', $
        'Z AXIS', resp

IFTHEN/ resp == 1
    JUMP/ BCK010:
ELSEIF/ resp == 2
    JUMP/ CANCEL:
ELSEIF/ resp == 5    $$ X AXIS
: (Option 1)
.
ELSEIF/ resp == 6    $$ Y AXIS
: (Option 2)
.
ELSEIF/ resp == 7    $$ Z AXIS
: (Option 3)
.
ENDIF

```

The interactive commands and the conditional branch statement (JUMP/) for the valid responses are as follows:

POS/ 'message', x, y, z, resp

JUMP/ back, cancel, not used, not used, pos def, resp

GPOS/ 'message', x, y, z, resp

JUMP/ back, cancel, OK, not used, pos def, resp

PARAM/ 'message', {'option' [,INT], variable}
[, ALTACTION, 'message',] resp

JUMP/ back, cancel, OK, alt–action, resp

IDENT/ 'message', ent list [,CNT,count] [,CURSOR, x, y, z]
[,MEMBER, {ON|OFF}], resp

JUMP/ back, cancel, OK, resp

TEXT/ 'message', string–variable, [ALTACTION,'message',]
resp [,DEFLT]

JUMP/ back, cancel, OK (no text), alt–action, OK (text), resp

CHOOSE/'string list', [DEFLT, n,] [ALTACTION, 'message',] resp

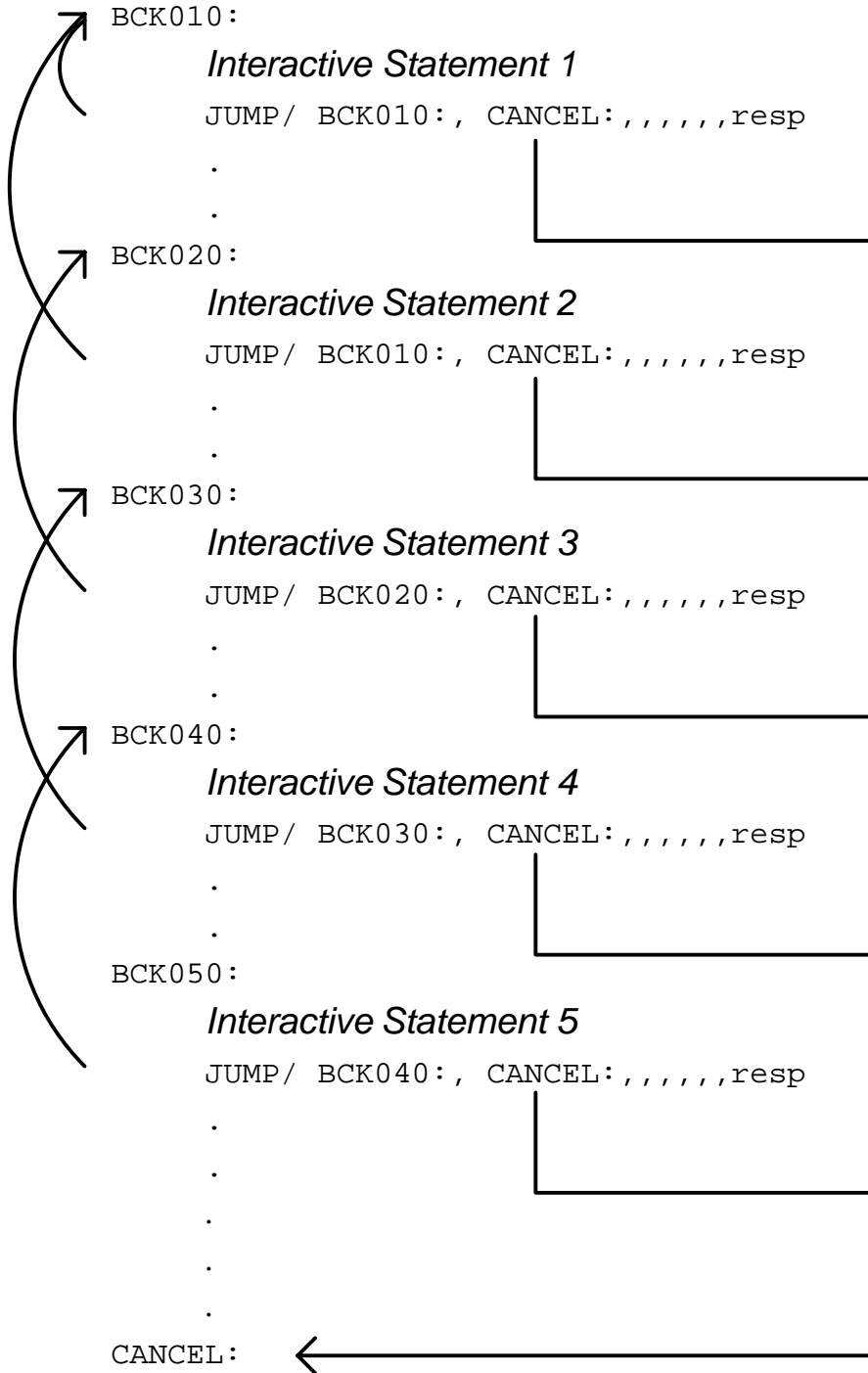
JUMP/ back, cancel, not used, alt–action, selection, resp

MCHOOS/ primary string, menu options, response array
[, ALTACTION, 'message',], response variable

JUMP/ back, cancel, OK, alt–action, resp



Program Outline using Interactive Statements



This example illustrates labels combined with the interactive programming using the outline described on the previous page.



Example Program with Interactive Commands

```

ENTITY/ hole(100)
NUMBER/ i, hole_radius, resp, x, y, z
i = 0
BCK010:
PARAM/ 'Specify hole radius', $
      'Radius', hole_radius, resp
JUMP/ BCK010:, CANCEL:, , resp
$$
$$ Check for negative or zero hole radius.
$$
IF/ HOLE_RADIUS,BCK010:,BCK010:,
BCK020:
POS/ 'Indicate hole location', $
     x, y, z, resp
JUMP/ BCK010:, CANCEL:, , , resp
$$
$$ Create hole.
$$
hole(i=i+1) = CIRCLE/ x, y, z, hole_radius
IFTHEN/ i <= 99
      JUMP/ BCK010:
ELSE/
      MESSG/'Max number of holes generated,',,$
          'terminating program.'
ENDIF
CANCEL:
      HALT

```

First Message

```

MESSG/TEMP,'CALCULATING SURFACE'
SCULPT=SSURF/PRIMA,L(1),L(3),CROSS,L(2),SPL

```

Second Message

```

MESSG/TEMP,'SURFACE CREATED'

```

Third Message

```

MESSG/'SEL EC TO CONTINUE'

```



The interactive commands prompt the user for input, temporarily stopping the GRIP program until a response is obtained. “Interact” means that you do one or more of the following:

- Press Back, Cancel, OK, or enter information on the dialog box.
- Select a screen position with the cross hairs on the graphics screen.
- Choose a method from the Generic Point Menu, then select a point.
- Set the values for numerical parameters.
- Select one or more objects on the graphics screen.
- Enter textual information.
- Choose an option (or options) from a menu on the dialog box.
- Read a message on the dialog box or graphics screen.

Table 6–2 Table of interactive commands

Command	Response Values Returned by the Interactive Command				
	1	2	3	4	5
POS	Back	Cancel	Not Used	Not Used	Pos Defined
GPOS	Back	Cancel	OK	Not Used	Pos Defined
PARAM	Back	Cancel	OK	Alternate Action	
IDENT	Back	Cancel	OK		
TEXT	Back	Cancel	OK (Return) No Text Entered	Alternate Action	OK (Return) Text Entered
CHOOSE	Back	Cancel	Not Used	Alternate Action	Menu Selection (5..18)
MCHOOS	Back	Cancel	OK	Alternate Action	

POS: Indicate Screen Position Point

Synopsis

POS/'message',x-coord,y-coord,z-coord,response

Description

This statement allows three coordinate values to be obtained by using the screen position indicator which will automatically appear upon execution of this statement.

CAUTION We do not recommend using the asterisk character (*) in menu prompts or menu options.

POS works with the **BACK** and **CANCEL** buttons. If **BACK** is selected during program execution while the **POS** statement is pending, the value of 1 is assigned to the response variable. **CANCEL** will cause 2 to be assigned. The values 3 and 4 are not used, and may not be used. Once the position is defined, the value of 5 will be assigned to the response variable and program execution will continue.

Below is a list of values assigned to the response variable:

Response	Assignment
Back	1
Cancel	2
Not Used	3
Not Used	4
Position Defined	5

'message'

A string which represents the title of the menu you want to display. The string can contain up to 40 characters.

x-coord,y-coord,z-coord

Three variables which will be assigned the coordinate values obtained from the screen position indicated by the user during program execution.

response

A variable which will be assigned a numerical value based on the user response.



Example

This example demonstrates the creation of a rectangle using the **POS** statement which prompts the user to indicate the diagonal corner screen position points.

Declarations

```

ENTITY/L(4)
BCK010:
    POS/'Define First Corner', $
        X1, Y1, Z1, RSP
    JUMP/BCK010:, CANCEL:,,, RSP
BCK020:
    POS/'Define Second Corner', $
        X2, Y2, Z2, RSP
    JUMP/BCK010:, CANCEL:,,, RSP

L(1)=LINE/X1, Y1, X2, Y1
L(2)=LINE/X2, Y1, X2, Y2
L(3)=LINE/X2, Y2, X1, Y2
L(4)=LINE/X1, Y2, X1, Y1

CANCEL:
    HALT
    
```

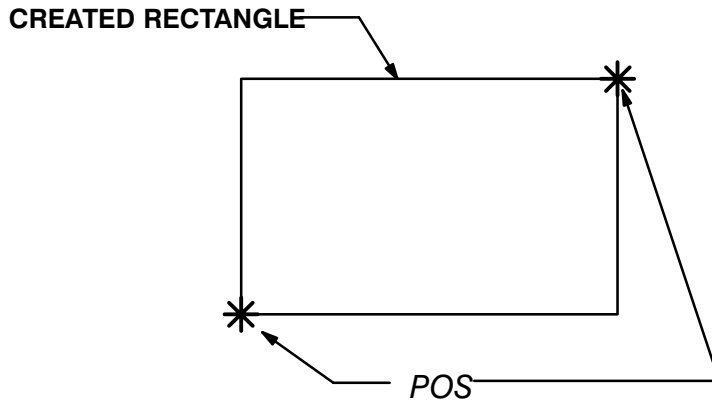


Figure 6–1 Use of the POS statement to create a rectangle

GPOS: Indicate Generic Point Position

Synopsis

GPOS/'message',x-coord,y-coord,z-coord,response

Description

The following allows three coordinate values to be obtained via the *Point Subfunction* menu. For more information on the *Point Subfunction* menu, refer to the *Unigraphics Essentials* manual.

CAUTION We do not recommend using the asterisk character (*) in menu prompts or menu options.

The **GPOS** statement works with the **Back**, **OK**, and **Cancel** options. If **Back** is selected during program execution while the GPOS statement is pending, the value of 1 is assigned to the response variable. **Cancel** will cause 2 to be assigned. **OK** will cause 3 to be assigned. The value of 4 is not used, and may not be used.

Once the position is defined, the value of 5 will be assigned to the response variable and program execution will continue.

Below is a list of values assigned to the response variable:

Response	Assignment
Back	1
Cancel	2
OK	3
Not used	4
Position Defined	5

'message'

A string which represents the title of the menu you want to display. The string can contain up to 40 characters.

x-coord,y-coord,z-coord

Three variables which will be assigned the coordinate values obtained from the generic point menu.

response

A variable which will be assigned a numerical value based on the user response.

Example

This example demonstrates the creation of a rectangle using the **GPOS** statement to prompt for the diagonal corner points.

Declarations

ENTITY/L(4)

BCK010:

GPOS/'Define First Corner',\$
X1,Y1,Z1,RSP

JUMP/BCK010:,CANCEL:,,,RSP

BCK020:

GPOS/'Define Second Corner',\$
X2,Y2,Z2,RSP

JUMP/BCK010:,CANCEL:,,,RSP

L(1)=LINE/X1,Y1,Z1, X2,Y1,Z1

L(2)=LINE/X2,Y1,Z1, X2,Y2,Z1

L(3)=LINE/X2,Y2,Z1, X1,Y2,Z1

L(4)=LINE/X1,Y2,Z1, X1,Y1,Z1

CANCEL:HALT

6

PARAM: Enter Parameters

Synopsis

PARAM'message',{**'option'** [,INT],variable,} +
[,**ALTACT**,'message',]response

6

Description

This statement creates a data entry menu which is displayed on the dialog box and allows the user to interactively assign numerical values to variables, which are associated with options in the menu.

CAUTION We do not recommend using the asterisk character (*) in menu prompts or menu options.

PARAM works with the **Back**, **OK**, **Alternate Action**, and **Cancel** buttons on the Unigraphics. If **Back** is selected during program execution while the **PARAM** statement is pending, the value of 1 is assigned to the response variable. **Cancel** will cause 2 to be assigned. **OK** will cause 3 to be assigned. **Alternate Action** will cause a value of 4 to be assigned.

You can create an **Alternate Action** option by using the minor word **ALTACT**, followed by a message, as the parameter just before the response variable. The **ALTACT** message displays at the bottom of the dialog box when the menu is displayed during statement execution. You can make the message up to forty characters long.

Below is a list of values assigned to the response variable:

Response	Assignment
Back	1
Cancel	2
OK	3
Alternate Action	4

'message'

A string which represents the title of the menu you want to display. The string can contain up to 40 characters.

'option'

Each option defined will be displayed on the menu created by the **PARAM** statement. The **PARAM** statement may contain a maximum of 14 options, each consisting of up to 15 characters. Options are truncated on some design stations.

INT

Minor word that indicates that the entered value must be an integer. This will be indicated in the menu by the absence of a decimal point in the assigned value.

variable

The variables which will be assigned the entered values. The **PARAM** statement may contain a maximum 14 variables; one for each of the options programmed.

ALTACT

Minor word that indicates that an alternate action message will be displayed as the last option in the dialog box. If **Alternate Action** is selected, the response variable will equal 4.

'message'

A string or string variable which represents the **ALTERNATE ACTION** message. The string can contain up to 40 characters.

response

A variable which will be assigned a numerical value based on the user response.

Example

This example demonstrates the use of the **PARAM** statement to create a user-defined menu.

Declarations

NUMBER/dia, rad, nholes

```
dia=.375
rad=4.5
nholes=8
```

BCK010:

```
PARAM/'Circular Hole Pattern',$
'Hole Diameter',dia,$
'Radius',rad,$
'No. of Holes',INT,nholes,RSP
```

```
JUMP/BCK010:,CANCEL:,,RSP
```

The previous program segment would display the following menu on the cue line and dialog box.

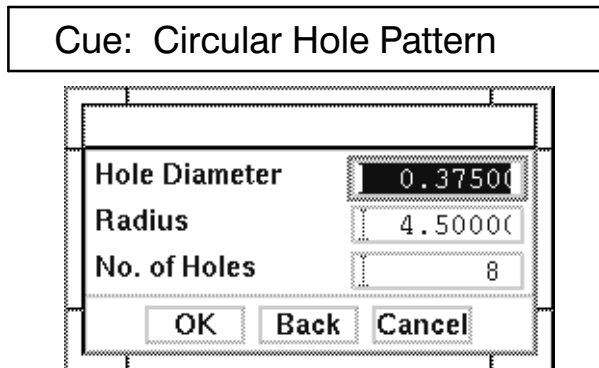


Figure 6–2 Menu displayed on the cue line and dialog box for the given example

NOTE

The last option above does not have a decimal point because the minor word **INT** was included before the variable in the **PARAM** statement.

Activity: Interactive L–Shape

6

Modify the program you wrote for Activity 1 to include the following interactive commands:

Follow these directions.

- Add an interactive command to allow the system user to enter the dimensions for this figure.

- Add an interactive command to allow the system user to indicate the coordinates of the lowest, leftmost corner of the figure. These coordinates will be used for the x and y values.

When you finish the assignment, demonstrate the program to your instructor.

Advanced Interactive Commands

Lesson 7

Objectives

- Demonstrate an understanding of the types of interactive commands available.
- Use interactive commands in conjunction with the branching and logic commands to build a user interface.



Interactive Commands

A variety of interactive commands are available in the programming language. These commands display menus on the dialog box and graphics screen for the user to enter information. Many of these commands provided the Back and Cancel options so that the user can reverse a previously made decision or exit the GRIP program.

Table 7-1 Advanced Interactive commands

Function	Format
Object selection	IDENT/ 'message', ent list [,CNT,count] [,CURSOR, x, y, z] [,MEMBER, {ON OFF}], resp
Obtain textural data	TEXT/ 'message', string-variable, [ALACT,'message',]resp [,DEFLT]
Select one item from a menu list	CHOOSE/'string list', [DEFLT, n,] [ALACT, 'message',] resp
Select multiple items from a menu list	MCHOOS/ primary string, menu options, response array [, ALACT, 'message',] resp



IDENT: Select Objects

Synopsis

IDENT'message'[,SCOPE,{WORK|ASSY|REF}],
obj_list[,CNT,count]
 [,CURSOR,x-coord,y-coord,z-coord]
 [,MEMBER,{ON|OFF}],response

Description

This statement allows objects selected by the user interactively to be placed in an object list.

IDENT works with the **Back**, **OK**, and **Cancel** options of Unigraphics. If **Back** is selected during program execution while the **IDENT** statement is pending, the value of 1 is assigned to the response variable. **Cancel** will cause 2 to be assigned. **OK** will cause 3 to be assigned.



CAUTION We do not recommend using the asterisk (*) in menu prompts or menu options.

Below is a list of response values assigned to the response variable:

Response	Assignment
Back	1
Cancel	2
OK	3

The **IDENT** statement will cause the system to display the *Class Selection Subfunction* dialog box. Also the 'message' will be displayed in the cue line. For more information on the *Class Selection Subfunction* refer to the Unigraphics Essentials manual.

NOTE

If the object list is a simple object variable or a single object of an object array (OBJ(1) of OBJ(10)), the class selection menu will not appear, but the crosshairs will be displayed and only a single object selection will be allowed.

message

A string which represents the title of the menu you want to display. The string can contain up to 40 characters.

SCOPE

Minor word which specifies the scope of object selection. **ASSY** is the default.

WORK

Allows you to select only objects which belong to the work part. This includes immediate components of the work part. If you select an object occurrence, the prototype is returned.

ASSY

Allows you to select any object or object occurrence in the assembly. No scope restrictions are applied.

REF

Allows you to select objects which belong to the work part or its subassembly. If you select an object occurrence, the prototype is returned.

obj list

An object list or array which will be assigned the selected objects.

NOTE

If a group is selected, the object list or array must be large enough to contain the group plus the objects in the group. For example, an object list or array of four would be necessary to hold a group consisting of three objects.

CNT

Minor word that indicates that a count of objects selected is desired.

count

A variable which will be assigned the number of objects selected.

CURSOR

Minor word that indicates that the coordinates of the screen position used for the selection of an object are to be retained.

x-coord, y-coord, z-coord

Three variables which will be assigned the coordinates of the screen position used for the selection of an object. The assigned value of these variables may be unpredictable if a technique other than cursor is used.



MEMBER

Minor word that indicates that the parameter in the following field controls the status of member selection. This option is most important when the user is not prompted with the *Class Selection Dialog* (where the member selection status could be changed). This option has an effect when single selection is used.

If the member parameter is not specified, the default is **ON** unless selection of either **groups** or **components** is enabled. If it is enabled, member selection is automatically **OFF**.

ON

Minor word that indicates that member selection is on. This allows you to select members of groups or components without selecting the group or component.

OFF

Minor word that indicates that member selection is off. When you select a member of a group or component, the entire group or component is selected.

response

A variable which will be assigned a numerical value based on the user response.



Example This example demonstrates the use of the **IDENT** statement to delete up to 25 user selected objects.

Declarations ENTITY/OBJ(25)
 BCK010:
 IDENT/'Select Objects to Delete',\$
 OBJ,CNT,NUM,RESP
 JUMP/BCK010;,LBL020;,,RESP
 DELETE/OBJ(1..NUM)
 LBL020:

Execution of the **IDENT** statement will cause the following *Class Selection Dialog* menu to be displayed.



Cue: Select Objects to Delete

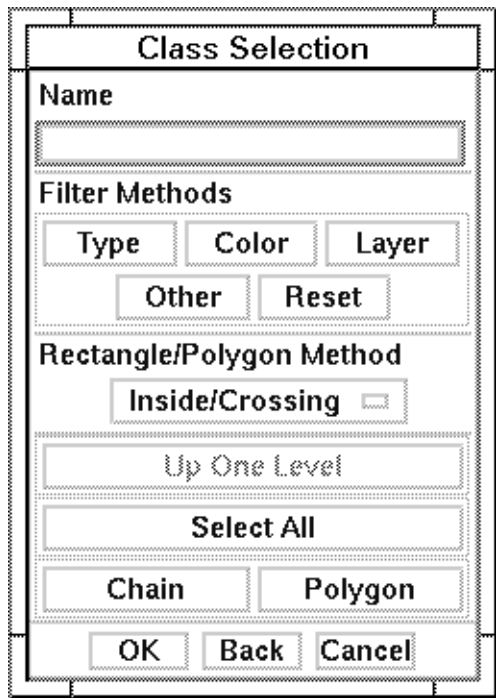


Figure 7–1 Class selection dialog

NOTE If the number of objects selected exceeds the object array size, the message “TOO MANY OBJECTS” will appear on the dialog box when the **OK** key is depressed. To continue, press the OK again. When this error occurs, the variables will be set as follows:

- The value returned in the response variable will be equal to zero(0).

- The count variable, if used, will reflect the number of objects that the user attempted to select.
- The object array passed to the **IDENT** statement will contain identifiers to selected objects. For example, consider the example above where an object array with a dimension of 25 is used. If the user attempts to select 100 objects, an error message will be displayed and the object array will contain the identifiers to the first 25 selected objects.



ALTACT

Minor word that indicates that an **ALTERNATE ACTION** message will be created on the dialog area during statement execution. If **ALTERNATE ACTION** is selected, the response variable will equal 4.

'message'

A string or string variable which represents the **ALTERNATE ACTION** message. The string can contain up to forty characters.

response

A variable which will be assigned a numerical value based on the user response.

DEFLT

Minor word that indicates that **OK** can be selected to accept text which has been previously assigned to the string variable specified. The contents of the string variables will be displayed by the **TEXT** statement. The user may either accept the text or type in new text. If the **DEFLT** Minor word is not specified, text which may have been assigned to the string variable will not be displayed. If **OK** or **Return** are used to accept the default text, the value of 3 will be assigned to the response variable.



Example This example demonstrates the use of the **TEXT** statement.

Declarations STRING/dwgno(30)

```
dwgno = '72A5001'          $$ Assign a default value
BCK010:
TEXT/ 'Enter Drawing Number', dwgno, resp, DEFLT
JUMP/ BCK010:, CANCEL:, resp

PRINT/ 'The Drawing Number is '+dwgno
```



The **TEXT** statement in the previous program segment would display the following menu. Notice the default text '72A5001' appears in the dialog box because the **DEFLT** word was used. If the **DEFLT** word is not used, the text entry area in the dialog box would be blank.

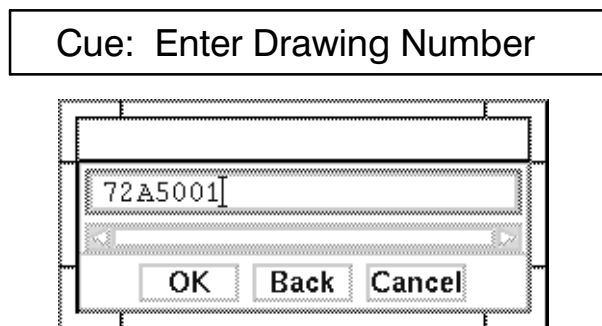


Figure 7–2 Menu displayed using the TEXT statement

CHOOSE: Choose Single Option

Synopsis

CHOOSE/'message',string
list,[DEFLT,n,][ALFACT,'message',]response

Description

This statement creates a menu of options which is displayed on the dialog box. One of the options may be selected from the menu. The option selected by the user causes a numerical value to be assigned to a variable called the *response variable*. This value may be used later in the program. The most common use is with a **JUMP** statement. This technique allows you to perform different operations in your program (by jumping to different labels based on the user response) based upon the option selected from the menu created by the **CHOOSE** statement.

CAUTION We do not recommend using the asterisk character (*) in menu prompts or menu options.

The **CHOOSE** statement works with the **Back**, **Cancel** and **Alternate Action** options within Unigraphics. **Back** will cause the value of 1 to be assigned to the response variable. **Cancel** will cause 2 to be assigned. The value 3 is not used, and may not be used. **Alternate Action** will cause 4 to be assigned to the response variable.

You can create an alternate action option by using the minor word **ALFACT**, followed by a message, as the parameter just before the response variable. The **ALFACT** message displays as the last option in the dialog area when the menu is displayed during statement execution. You can make the message up to forty characters long.

The response values 1–4 are reserved for the **Back**, **Cancel**, and **Alternate Action** selections. Therefore, the first value available for assignment from the menu options created by the **CHOOSE** statement is 5. The first option on the created menu will return 5, the second option will return 6, the third, 7, and so on. There is a maximum of fourteen options on a menu. The following is a



list of values assigned to the response variable for the **CHOOSE** statement:

Response	Assignment
Back	1
Cancel	2
Not used	3
Alternate Action	4
Option #1	5
Option #2	6
.	.
.	.
Option #14	18



string list

A list of strings or string variables which constitute the **CHOOSE** menu. The first parameter in the list is the message which is covered in the beginning of this section and the remaining parameters in the list will be used as options. Each string can contain up to 40 characters. Using a hyphen “-” for one of your options will create a nonselectable separator line.

NOTE Since the maximum number of options in a choose menu is 15, the number of parameters in the list may not be greater than 15.

DEFLT

Minor word that indicates that a menu default position is desired.

n

The item number in the menu where the default caret will be displayed.

ALTACTION

Minor word that indicates that an alternate action message will be create on the dialog area during statement execution. If **Alternate Action** is selected, the response variable will equal 4.

'message'

A string or string variable which represents the alternate action message. The string can contain up to 40 characters.

response

A variable which will be assigned a numerical value based on the user response.

Example

This example demonstrates the use of the **CHOOSE** statement to create a menu to prompt the user for a standard object.

```

BCK010:
CHOOSE/'Select Standard Part',$
      'Bolt',$
      'Nut',$
      'Washer',DEFLT,1,RSP
JUMP/BCK010:;CANCEL:;, , ,LBL010:;LBL020:;LBL030:;RSP

LBL010:                                $$ option 1
      ptype='BOLT'
      JUMP/LBL040:

LBL020:                                $$ option 2
      ptype='NUT'
      JUMP/LBL040:

LBL030:                                $$ option 3
      ptype='WASHER'
LBL040:
    
```



The previous example programs would display the following cue line and dialog box:

Cue: Select Standard Part

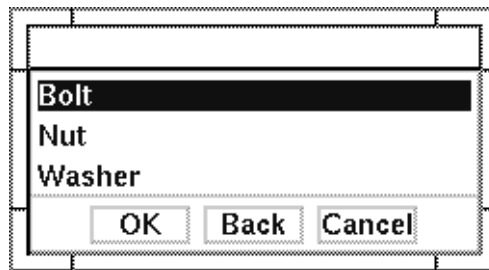


Figure 7–3 The displayed choose statement.

Choose Options (Cont'd)

Example The program below is a simple example of use of the CHOOSE/ statement.

```

Example                    ENTITY/ ln
                           NUMBER/ resp,x,y,z
BCK010:
                           GPOS/'INDICATE POSITION',$
                           x,y,z,resp
                           JUMP/BCK010:,CANCEL:,,,resp

                           ln=LINE/x,y,z,x-5,y,z

BCK020:
                           CHOOSE/'IS LINE OK?',        $        $$ Main title
                           'YES, CONTINUE', $        $$ Option 1
                           'NO',                    $        $$ Option 2
                           DEFLT,1,resp
                           JUMP/BCK020:,BCK020:,,,LBL030:,,,resp
                           DELETE/ln
                           JUMP/BCK010:                $$ Re-indicate position
LBL030:
                           &COLOR(ln)=&RED
CANCEL:
                           HALT
    
```

Frequently, in applications which create geometry, it is necessary to allow the user to confirm that the geometry has been created properly.

In the example above, the user must select either option 1, YES, CONTINUE, or option number 2, NO, from the menu. If the user selects BACK or CANCEL, the menu will simply be redisplayed.



MCHOOS: Choose Multiple Options

Synopsis

**MCHOOS/primary string, menu options, response array,
[ALFACT,'message',]response variable**

Description

This statement creates a dialog box with a maximum of 14 options (15 options with the Alternate Action option). You may then interactively choose any or all of these options from the menu. Each option you choose causes a numerical value to be assigned to a variable, called the *response array*. You must choose **OK** to enter the chosen options into the response array.

CAUTION We do not recommend using the asterisk character (*) in menu prompts or menu options.

The **MCHOOS** statement works with the **Back**, **Cancel** and the alternate action option. **Back** will cause the value of 1 to be assigned to the response variable. **Cancel** will cause 2 to be assigned. The value 3 indicates that zero or more menu options were chosen along with **OK**. The alternate action option will cause 4 to be assigned to the response variable.

You can enable an alternate action option by using the minor word **ALFACT**, followed by a message, as the parameter just before the response variable. The **ALFACT** message displays at the bottom of the dialog box when the menu is displayed during statement execution. You can make the message up to forty characters long.

The response array entries will contain a 1 if the menu option that matches the entry's index is chosen, and a 0 otherwise. For example, if options 1 and 3 are chosen from a five option menu, the response array would contain 1,0,1,0,0.



There is a maximum of fourteen options on a menu. Below is a list of response values assigned to the response variable for the **MCHOOS** statement:

Response	Assignment
Back	1
Cancel	2
OK	3**
Alternate Action	4**

NOTE ** **OK** or **Alternate Action** indicates that one or or more menu selections were made.

primary string

A string or string variable consisting of the title of the menu, which can be up to 40 characters.

menu options

A string list consisting of up to 14 menu options. Each string can contain up to 40 characters, although the display may be truncated by the window system.

NOTE The string list may only contain 15 items (the menu title and 14 options).

response array

An array which handles the responses from the chosen options. You may use a subrange; however, it must be large enough to handle every menu option. The number of responses should match the number of menu options. The system returns a value of 1 if the corresponding option is chosen, a value of 0 if the option is not chosen.

ALTACTION

Minor word that indicates that an **ALTERNATE ACTION** message will be created on the menu during statement execution. If the **ALTERNATE ACTION** option is selected, the response variable will equal 4.

'message'

A string or string variable which represents the **ALTERNATE ACTION** message. The string can contain up to 40 characters.



response variable

A variable which will be assigned a numerical value based on the user response.



Activity: Interactive Sheet Metal Part

Modify the program written for Activity 2. Follow these steps:

- The program is to be modified to permit construction of any rectangle given the width and height dimensions. Add an interactive command to allow the user to enter the width and height dimensions for this figure from the keyboard.
- Add an interactive command to allow the user to enter the values for the coordinates of the center of the rectangle using the generic point menu.
- Add an interactive command to allow the user to choose whether he wants the corners filleted or not.
- If the user wants the corners filleted, add an interactive command to allow the radii of these fillets to be entered from the keyboard.

In order to allow clean construction of the figure, all interactive commands must come before any geometry construction commands.

When you finish the assignment, demonstrate the program to your instructor.

Optional: Add logical statements to check the validity of the data the user has entered. This will provide cleaner construction of the part. For example, a negative or zero radius fillet value should not be allowed. Also, the fillet radius should be less than the dimensions of the figure.

Solid Object Modeling Commands

Lesson 8

Objectives

- Demonstrate an understanding of the commands for creating solids.
- Perform various operations on solids.



Solid Feature Creation

This section contains the GRIP statements necessary to create solid bodies. This subset of commands include: block, cylinder, solid of revolution, and extruded solid. Many of the solid creation statements in GRIP create parametric solid bodies. Any creation parameters (e.g., dimensions of a block) are stored with the solid body and may be edited. Usually the resulting body is associative with any geometry used to construct the body.

Table 8–1 Solid creation statements

Function	Format
Solid Block creation	SOLBLK/ORIGIN,xc,yc,zc, SIZE,dx,dy,dz [,IFERR,label:]
Solid Cylinder creation	SOLCYL/ORIGIN,xc,yc,zc, HEIGHT,h,DIAMTR,d [,AXIS,i,j,k] [,IFERR,label:]
Solid Extrusion	SOLEXT/obj_list,HEIGHT,h [,AXIS,i,j,k] [,IFERR,label:]
Revolved Solid	SOLREV/obj_list, ORIGIN,xc,yc,zc, ATANGL,a [,AXIS,i,j,k] [,IFERR,label:]

NOTE

When using solid feature statements on assemblies, you need to make sure that the operation (or creation statement) is performed on objects in the work part. If the objects are not in the work part, the statement is not executed and an error is reported.



<i>Name</i>	SOLBLK	Solid Block
<i>Synopsis</i>	obj = SOLBLK/ORIGIN,xc,yc,zc,SIZE,dx,dy,dz[,IFERR,label:]	
<i>Description</i>	Allows you to create a parametric solid block of any size. The origin is on a corner of the block. The edges of the block are aligned with the axes of the WCS.	
<i>Parameters</i>	ORIGIN Minor word indicating that the following values specify the block's origin. xc,yc,zc Work coordinates which specify the origin of the block. SIZE Minor word indicating that the following values specify the size of the block. dx ,dy, dz Three length values for the block's edges. These values can be positive or negative, and determine the placement of the block relative to its origin. A positive value causes the system to create that particular edge in the positive direction of the corresponding axis. A negative value causes the system to create that edge in the negative direction of the corresponding axis. IFERR,label: Specifies a label to which program execution jumps if an error occurs. Possible errors include an invalid origin specification, invalid parameter value, etc.	

Example This example demonstrates the creation of two solid blocks, both possessing the same origin point. One has positive edge length values, the other negative edge lengths.

Declarations ENTITY/BLOCK1,BLOCK2

Geometry Definition BLOCK1 = SOLBLK/ORIGIN,1,3,2,SIZE,1,2,.5
BLOCK2 = SOLBLK/ORIGIN,1,3,2,SIZE,-1,-2,-.5

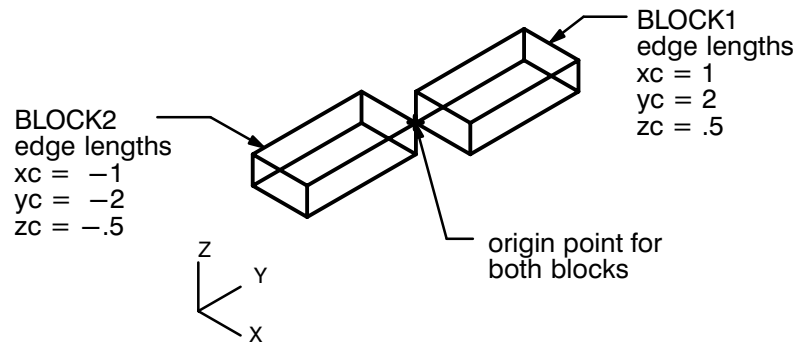


Figure 8–1 Solid blocks created using positive vs. negative edge lengths

<i>Name</i>	SOLCYL Solid Cylinder
<i>Synopsis</i>	obj = SOLCYL/ORIGIN,xc,yc,zc,HEIGHT,h,DIAMTR,d [,AXIS,i,j,k][,IFERR,label:]
<i>Description</i>	Allows you to create a parametric solid cylinder by specifying its origin, height, and diameter.
<i>Parameters</i>	<p>ORIGIN Minor word indicating that the following coordinate values specify the cylinder origin.</p> <p>xc,yc,zc Work coordinate values which specify the origin of the cylinder. The origin is located on one of the circular faces of the cylinder.</p> <p>HEIGHT Minor word indicating that the following value specifies the cylinder height.</p> <p>h The distance between the two circular faces of the cylinder. The height can be positive or negative and determines one of two possible cylinders.</p> <p>DIAMTR Minor word indicating that the following value represents the cylinder diameter.</p> <p>d The diameter of the cylinder.</p> <p>AXIS Optional minor word indicating that the following values specify the cylinder axis.</p> <p>i, j, k Specifies a vector which defines the axis of the cylinder. The system defines the axis parallel to this vector. If none is specified, the system defines the cylinder axis parallel to the ZC axis of the WCS.</p>

IFERR,label:

Specifies a label to which program execution jumps if an error occurs. Possible errors include an invalidly specified axis or origin, parameter values not within the correct range, etc.

Example

This example demonstrates the creation of two cylinders, both with the same center. One cylinder (CYL1) is created with a positive height value, the other (CYL2) with a negative height value. The axis for both is specified in the positive Z direction.

Declarations

ENTITY/CYL1,CYL2

Geometry Definition

CYL1=SOLCYL/ORIGIN,2,1,1,HEIGHT,1.5,DIAMTR,.5
 CYL2=SOLCYL/ORIGIN,2,1,1,HEIGHT,-2,DIAMTR,1,AXIS,0,0,1

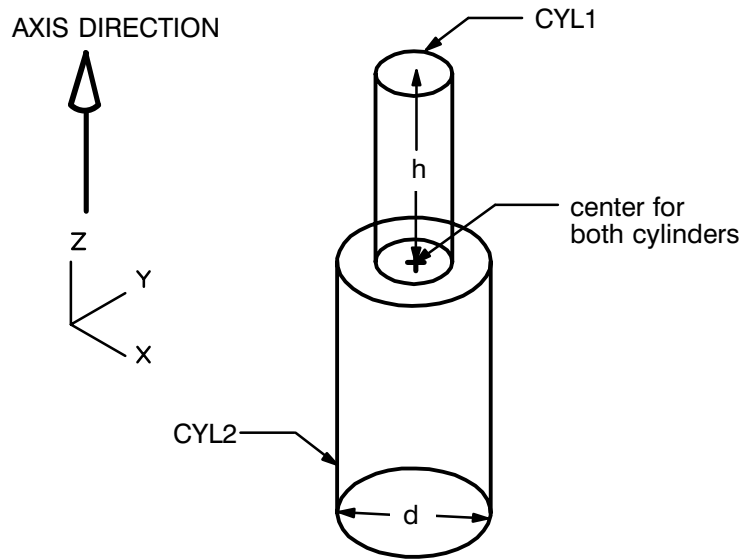


Figure 8–2 Cylinder creation; positive vs. negative height values



<i>Name</i>	SOLEXT Extruded Solid
<i>Synopsis</i>	<u>obj</u> = SOLEXT/<u>obj list</u>,HEIGHT,h[,<u>AXIS</u>,i,j,k] [,IFERR,label:]
<i>Description</i>	<p>Allows you create a parametric solid body by sweeping a series of curves a specified distance in a specified direction. The specified curves may form an open or closed boundary. Be sure to specify the curves in consecutive order (i.e., do not skip an adjacent curve and then try to specify it later).</p> <p>You may also create an extruded solid with holes in it. It makes no difference whether you first define the outer boundary or the hole.</p>
<i>Parameters</i>	<p><u>obj list</u> A list of curves to be extruded along the specified axis. If the specified curves are not closed, the system creates a solid sheet. A closed loop of curves creates a solid. The curves do not have to be coplanar. The maximum number of objects allowed is 514.</p> <p>NOTE In the case of a closed loop of curves, if the curves are not coplanar, the system attempts to attach a sheet body to the loop. If no sheet can be attached, the extrusion fails. Planar, spherical, conical, cylindrical, and toroidal bodies can be attached.</p> <p>HEIGHT Minor word indicating that the next value specifies the height of the extruded solid.</p> <p>h The distance that the specified curves sweep in the specified direction. Values can be positive or negative. Positive values sweep in the direction of the axis vector, negative values sweep in the opposite direction of the axis vector.</p> <p>AXIS Optional minor word indicating that the next values specify the axis for the extruded solid.</p> <p>i,j,k Specifies a vector which defines the direction for the sweep.</p>

IFERR,label:

Specifies a label to which program execution jumps if an error occurs. Possible errors include an invalidly specified axis, parameter values not within correct ranges, number of objects exceeds maximum, invalid objects listed, the system was unable to find a body while attaching a sheet to a face, the geometry fails to pass checks while subdividing a face, etc.

Example

This example demonstrates the creation of an extruded solid from a series of curves. The program allows the curves to be selected interactively, then sweeps them along the ZC axis at a distance of 2.

Declarations

```
ENTITY/OBJ(100),SOLID1
BCK010:
    IDENT/'PICK CURVES',OBJ,CNT,NUM,RSP
JUMP/BCK010:;CANCEL:;,RSP
```

Geometry Definition

```
SOLID1 = SOLEXT/OBJ(1..NUM),HEIGHT,2,AXIS,0,0,1
CANCEL:
    HALT
```

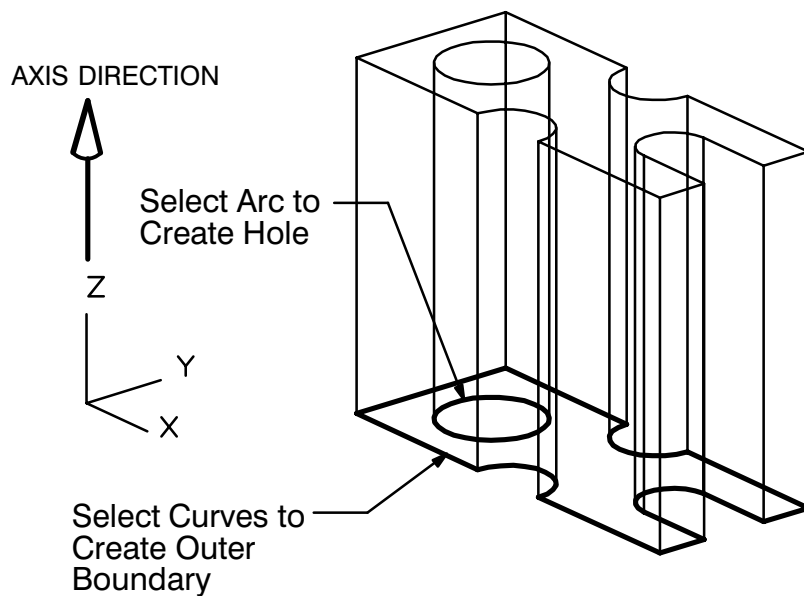


Figure 8–3 Curves Swept Along ZC Axis to Form Extruded Solid

<i>Name</i>	SOLREV Solid of Revolution
<i>Synopsis</i>	obj = SOLREV/obj_list,ORIGIN,xc,yc,zc,ATANGL,a[,AXIS,i,j,k] [,IFERR,label:]
<i>Description</i>	Allows you to create a solid of revolution by revolving a series of curves around a specified axis. The resulting solid is parametric. The specified curves may form an open or closed boundary. Self intersecting solids are not allowed.
<i>Parameters</i>	<p>obj list A series of curves to be revolved around the specified axis. If the specified curves are not closed, the system creates a solid sheet. A closed loop of curves creates a solid. The curves do not have to be coplanar. The maximum number of objects allowed is 514.</p>

NOTE In the case of a closed loop of curves, if the curves are not coplanar, the system attempts to attach a sheet body to the loop. If no body can be attached, the extrusion fails. Planar, spherical, conical, cylindrical, and toroidal bodies can be attached.

ORIGIN

Minor word indicating that the following values specify the origin for the axis of rotation.

xc,yc,zc

Work coordinates specifying the origin of the solid.

ATANGL

Minor word indicating that the following value specifies the angle of rotation.

a

The angle of rotation for the solid. The positive direction is determined by the right hand rule for rotation. Align the thumb of your right hand with the displayed vector. The direction in which your fingers curl is the direction of positive rotation.



AXIS

Minor word indicating that the following values specify the axis of rotation.

i, j, k

Specifies a vector which defines the axis of rotation. The system defines the axis as parallel to the specified vector, passing through the origin. If no axis is specified, the YC axis is used.

IFERR,label:

Specifies a label to which program execution jumps if an error occurs. Possible errors include: an invalidly specified axis or origin, parameter values not within correct ranges, number of objects exceeds maximum allowed, invalid object types listed, the geometry fails to pass checks while subdividing a face, the system is unable to find a body while attaching a sheet to a face, etc.

Example

This example demonstrates the creation of a solid of revolution from a series of curves. The program allows the curves to be selected interactively, then revolves them around the XC axis at an angle of 180°.

Declarations

```
ENTITY/OBJ(100),SOLID1
BCK010:
    IDENT/'PICK CURVES',OBJ,CNT,NUM,RSP
JUMP/BCK010:, CANCEL:,,RSP
```

Geometry Definition

```
SOLID1=SOLREV/OBJ(1..NUM),ORIGIN,2,2,0,$
    ATANGL,180,AXIS,1,0,0
CANCEL:
    HALT
```



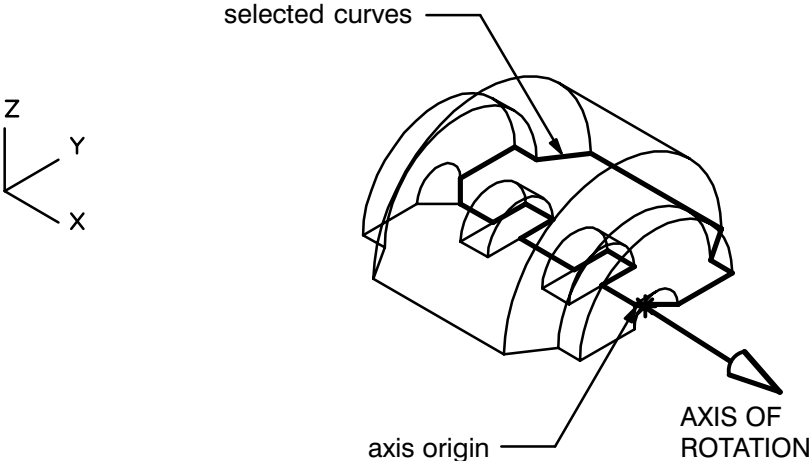


Figure 8–4 Solid of revolution creation



Solid Operations

Existing solids can be altered using GRIP commands. Blends, splits, and boolean operations can be performed on solids. Some of the operation statements result in a non-parametric body. Association between the original bodies and the construction curves (or sheets) may also be lost. The description for each statement explains whether or not the resulting solid body is parametric.

Table 8–2 Solid operation statements

Function	Format
Unite Solids	UNITE/ <u>obj</u> ,WITH, <u>obj list1</u> [,CNT,c][,IFERR,label:]
Subtract Solids	SUBTRA/ <u>obj</u> ,WITH, <u>obj list</u> [,CNT,c][,IFERR,label:]
Intersect Solids	INTERS/ <u>obj</u> ,WITH, <u>obj list1</u> [,CNT,c][,IFERR,label:]
Blend/Chamfer	BLEND/ <u>obj</u> ,{RADIUS CHAMFR},num [, <u>obj list1</u>][,VERT, <u>obj list2</u>] [IFERR,label:]

<i>Name</i>	UNITE Unite
<i>Synopsis</i>	<u>obj list</u> = UNITE/<u>obj</u>,WITH,<u>obj list1</u>[,CNT,c][,IFERR,label:]
<i>Description</i>	<p>Allows you to combine several solids into one by specifying a <i>target</i> solid and a list of solids to be united to the target. The resulting solid body is parametric. In order for the unite operation to be useful, the specified solids must intersect.</p> <p>In order to unite sheet bodies, they must meet at their edges unless they have overlapping faces.</p>
<i>Parameters</i>	<p><u>obj</u> The target solid.</p> <p>WITH Minor word specifying that the target solid is united with the following object list.</p> <p><u>obj list1</u> List of solid objects to be united with the target solid. Maximum allowed in list is 1000.</p> <p>CNT Optional minor word indicating that a count of solid objects is to be returned.</p> <p>c Variable which returns the number of solid objects.</p> <p>IFERR,label: Specifies a label to which program execution jumps if an error occurs. Possible errors include: maximum amount of objects exceeded, illegal object specified, cannot unite opposed sheets, can't unite solid with sheet, etc.</p>

Name **SUBTRA** Subtract

Synopsis **obj list = SUBTRA/obj,WITH,obj list[,CNT,c][,IFERR,label:]**

Description Allows you to subtract solids from solids, sheet bodies from sheet bodies, and sheet bodies from solids, as long as the sheet body cuts the solid completely. When a single body results, the body is parametric. When multiple bodies are generated, all the bodies are non-parametric.

Parameters **obj**
The target solid.

WITH

Minor word indicates that the following list of objects is to be subtracted from the target solid.

obj list1

A list of tool solids you wish to subtract from the target solid. Maximum number allowed in list is 1000.

CNT

Optional minor word indicating that a count of solid objects is to be returned.

c

Variable which returns the number of solid objects.

IFERR, label:

Specifies a label to which program execution jumps if an error occurs. Possible errors include: non-manifold body or boundary, maximum number of tool objects exceeded, illegal object type in list, etc.



<i>Name</i>	INTERS Intersect
<i>Synopsis</i>	<u>obj list</u> = INTERS/<u>obj</u>,WITH,<u>obj list1</u>[,CNT,c][,IFERR,label:]
<i>Description</i>	Allows you to create a separate solid from those portions shared by multiple solids. You can intersect solids with solids, sheet bodies with sheet bodies, and a solid with a sheet body, however, you cannot mix solids and sheets in the object list of tool solids. When a single body results, the body is parametric. When multiple bodies are generated, all the bodies are non-parametric.
<i>Parameters</i>	<p><u>obj</u> The target solid.</p> <p>WITH Minor word indicates that the target solid is to be intersected with the following list of solids.</p> <p><u>obj list1</u> A list of tool solids you wish to add to the target solid. Objects must be similar (all solids or all sheets). Maximum allowable number is 1000.</p> <p>CNT Optional minor word indicates that a count of solid objects is to be returned</p> <p>c Variable which returns the number of solid objects.</p> <p>IFERR, label: Specifies a label to which program execution jumps if an error occurs. Possible errors include: mixture of sheet and solid tool bodies, maximum number of tool solids exceeded, illegal object type in list, etc.</p>

<i>Name</i>	BLEND	Blend/Chamfer
<i>Synopsis</i>	BLEND/obj , { RADIUS CHAMFR }, num [, obj_list1] [, VERT , obj_list2] [IFERR , label :]	
<i>Description</i>	<p>Allows you to modify a solid by either rounding (blend) or beveling (chamfer) specified edges of the solid.</p> <p>The BLEND statement works in much the same way as the Blend option in the Modeling application except that you may only do fixed radius blends. See the <i>Modeling</i> manual.</p>	
<i>Parameters</i>	<p>obj The solid to be modified.</p> <p>RADIUS Minor word indicating that a blend operation is to be performed on the solid.</p> <p>CHAMFR Minor word indicating that a chamfer operation is to be performed on the solid.</p> <p>num When used with the RADIUS minor word, num is the positive radius of the blend. When used with the CHAMFR minor word, num is the offset distance between the intersection of the specified edges. When one edge is curved, the offset is measured along the path of the curved edge.</p> <p>obj_list1 First list of point objects which identify the edges of the solid to blend/chamfer. The points identify the edges by distance. If neither list of points is specified, then the system blends/chamfers every edge in the solid.</p> <p>VERT Optional minor word specifies that the following list of points identifies vertices on the solid.</p>	

obj list2

Second list of point objects which identify vertices on the solid. The points identify the vertices by distance. The edges involved in these vertices are blended/chamfered. If you do not specify either list of points, then the system blends/chamfers every edge in the solid.

IFERR,label:

Specifies a label to which program execution jumps if an error occurs. Possible errors include: illegal object specified, radius values must be positive, etc.



Solid Object EDAs

The following table describes the EDAs that can be used specifically on Solid Objects. The &SOLDAT command is generally used in conjunction with the SOLENT/ command. The Edge Type command will indicate the wireframe entity type of the underlying curve that represents an edge. Solid density can be set for individual solids. The GPA &SDENS will set the default solid density for new solids.

Function	EDA Symbol	Access Type	Data Type
Number of Faces or Edges/	&SOLDAT(obj,{FACE EDGE}{[,IFERR,label:]})	R/O	Number >= 0
Determine Edge Type	&EDGTYP(obj[,IFERR,label:])	R/O	Number [0,3,5,6,9]
Solid Density	&SDENS(obj)*	R/W	Number >=0

* This command is also a GPA.

8

Solid Object GPAs

The following table describes the GPAs that can be used to set the preferences for solid object display and creation. The density units can be read/set using the numbers or GPA constants: 1 = &LBIN, 2 = &LBFT, 3 = &GCM, 4 = &KGM.

Function	GPA Symbol	Access Type	Data Type
Solid Density	&SDENS*	R/W	Number >=0
Density Units	&SDUNIT	R/W	Number [1..4]

* This command is also an EDA.

Activity: Extrude the L–Shape

The L–Shape geometry created in a previous exercise can be used with the solid creation commands to generate a solid body. Use the geometry created in that program to create an extruded solid with a hole.

- Copy the L–shape program into a new file. Do not forget to use your initials in the file name.
- Extrude the lines in the +Z direction using a distance of `ext_ht=4`.
- Put a 1 inch horizontal hole through the center of the long portion of the L. This will require a boolean subtraction operation. You will need to create a cylinder at the appropriate location and orient the cylinder in the XC direction. The location of the hole should be calculated from the reference x,y location.

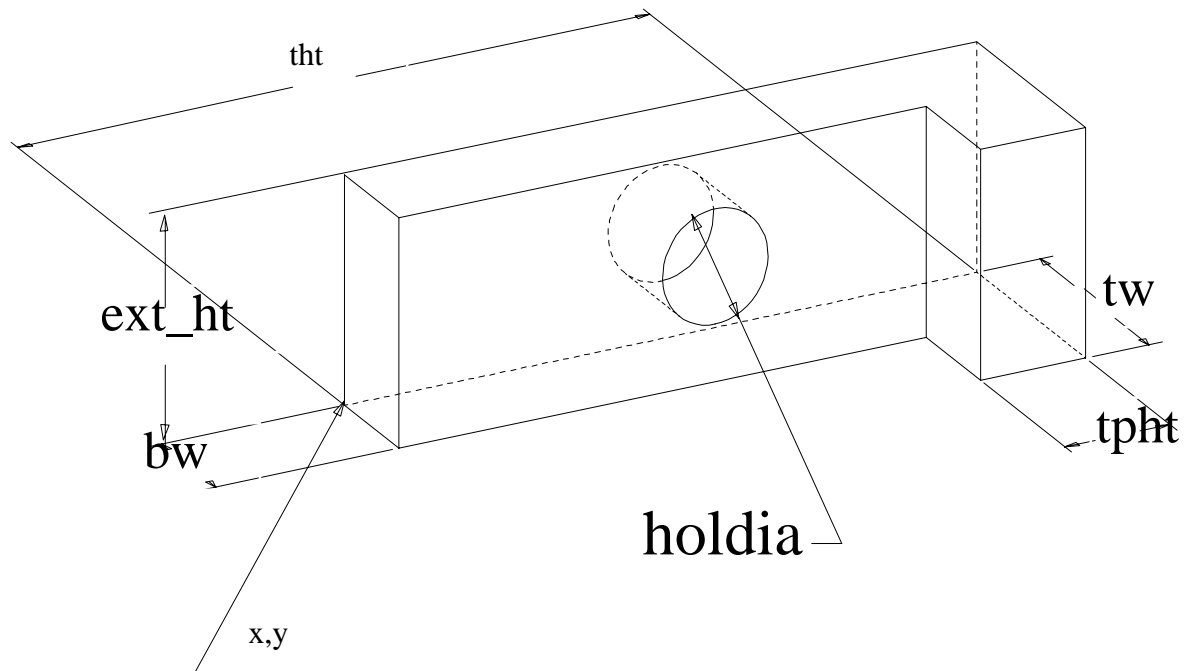


Figure 8–5 Extruded L–Shape

- Additional Challenge:** Add prompts for the extrusion distance and diameter. Error check the new data.

Activity: Solid L–Shape

Another method to create the L–Shape is to start with solids rather than curves. In this activity, you will create a new program to generate the L–Shape from solids.

- Create a new GRIP source file using your initials in the file name.
- Use the size data for the L–Shape curve creation program and the extrusion data from the last exercise.

Variable	Value	Definition
x	0	X coordinate of lowest, leftmost corner of figure.
y	–3	Y coordinate of lowest, leftmost corner of figure.
bw	1	Bottom width of figure
tw	2	Top width of figure
tht	6	Total height of figure
tpht	2	Top height of figure
ext_ht	4	Extrusion distance
holdia	1	Diameter of the hole

- Put a 1 inch hole through the center of the long portion of the L. This will require a boolean subtraction operation. You will need to create a cylinder at the appropriate location and orient the cylinder in the XC direction. The location of the hole should be calculated from the reference x,y location.

TIP When you create the solid blocks for the L–shape, you should consider that they will be united. You can only unite objects that overlap or touch. When creating the solids for uniting, it is usually a better practice to overlap. If the solids are not touching within tolerance, a Non–manifold Solid error message will be displayed in a pop–up window.

Database Cycling

Lesson 9

Objectives

- Understand File Access.
- Examine geometric objects in the data model.
- Examine non–geometric objects in the data model.
- Learn reading and writing to text files.
- Understand printing data to the Listing Device.



File Access

This section covers the GRIP statements used to control access to Unigraphics and scratch files. Many of these statements are imbedded in Unigraphics, such as creating, retrieving, and filing a part.

Table 9–1 File Access Commands

Function	Format
Open a File	FETCH/{PART,'filespec' TXT,file#,'filespec'} [,IFERR,label:]
Create a File	CREATE/{PART,'filespec' {,INCHES MMETER} TXT,file# [, <u>number list</u>][,'filespec']},[,IFERR,label:]
Save a File	FILE/{PART TXT[,file#][,'filespec']},[,LINNO] [,IFERR,label:]
Close a File	FTERM/{PART[options] TXT,file#} [,IFERR,label:]
Delete a File	FDEL/'filespec'[,IFERR,label:]

NOTE When accessing scratch files, be aware that GRIP NC CLS files also use scratch file 1. If you attempt to write to scratch file 1 when it already contains a CLS file, the CLS file will be lost.

<i>Name</i>	FETCH	Open a File
<i>Synopsis</i>	FETCH/{PART,'filespec' TXT,file#,'filespec'}[,IFERR,label:]	
<i>Description</i>	This statement retrieves a copy of either a <i>part</i> file or a <i>text</i> file.	
<i>Parameters</i>	PART	Minor word that indicates that the file to be retrieved is a part file.
	TXT	Minor word which indicates that the file to be retrieved is a text file. The file pointer is left on the last line of the file after it is retrieved. This can be an important consideration if you should try to read that file.
	file#	The number of the scratch file into which the specified file is to be retrieved. Ten scratch file areas are available, therefore, the specified number must be between 1 and 10. The scratch file number is required for text files only. The file number cannot be used if the file is a part file.
	'filespec'	The filespec parameter is used to specify the name of the part or text file. The filespec is required for part or text files only.
	IFERR,label:	Specifies a label to which the program jumps if an error occurs.

<i>Name</i>	CREATE Create a File
<i>Synopsis</i>	CREATE/{PART,'filespec'[,INCHES MMETER] TXT,file# [, 'filespec']} [,IFERR,label:]
<i>Description</i>	<p>The following statement creates either a <i>part</i> file or a <i>text</i> file in one of the 10 scratch file areas.</p> <p>If you are creating a <i>part</i> file, a <i>filespec</i> is required.</p> <p>If you are creating a <i>text</i> file, the <i>filespec</i> is optional. A <i>filespec</i> is required to file the text.</p> <p>NOTE If you are creating a part file, the new part will contain all customer default parameter values. Any GPAs defined before the CREATE/PART statement will be reset to the current customer default value.</p>
<i>Parameters</i>	<p>PART Minor word that indicates that a new part file is to be created.</p> <p>'filespec' The <i>filespec</i> associated with part file creation.</p> <p>INCHES,MMETER The units of measure, inches and millimeters respectively, which will be used for the part geometry.</p> <p>TXT Minor word indicating that a text file is to be created in one of the scratch file areas which are basically used as temporary working areas.</p> <p>file# The number of the scratch file in which the new file is to be created. Ten scratch file areas are available, therefore, the specified number must be between 1 and 10.</p> <p>'filespec' The <i>filespec</i> associated with text file creation. The <i>filespec</i> for a text file is optional in the create statement because it may be specified later in the FILE statement, if the file is to be saved.</p> <p>IFERR,label: Specifies a label to which the program jumps if an error occurs.</p>



<i>Name</i>	FILE Save a File
<i>Synopsis</i>	FILE/{PART TXT[,file#][,filespec']}[,LINNO] [,IFERR,label:]
<i>Description</i>	This statement saves a copy a <i>part file</i> or a <i>text file</i> without terminating the part or scratch file.
<i>Parameters</i>	<p>PART Minor word that indicates that the file to be saved is the current work part file.</p> <p>TXT Minor word that indicates that the file to be saved is the text file which is currently in the specified scratch file.</p> <p>NOTE The line pointer will reset to the top of the file after it has been saved.</p> <p>file# The number of the scratch file which is to be saved. This parameter may be used only if the file is a text file. Ten scratch file areas are available, therefore, the specified number must be between 1 and 10.</p> <p>'filespec' The optional filespec parameter. If a filespec is not specified the file will be saved with the name with which it was created or fetched. If you specify an existing file, then a "File already exists" message occurs.</p> <p>NOTE If a system error log or system log was fetched, the scratch file has no name and, therefore, one must be specified in the FILE statement.</p> <p>LINNO Minor word which indicates that the text file should be filed with the editor's line numbers</p> <p>IFERR,label: Specifies a label to which the program jumps if an error occurs.</p>

Name **FTERM** Close a File

Synopsis **FTERM/{PART[options] | TXT,file#}[,IFERR,label:]**

Description This statement closes an active part(s) or scratch file without saving. Options for the **PART** minor word allow you to either: close all parts or specify a part file name, close all the subassemblies of the part, or close modified part(s) without an error. If you do not use any options with the **PART** minor word, then the command closes the default work part.

“[options]” is equivalent to: **[,all] [,string] [,asmby] [,always]**. If you specify the **all** option, then the use of **string**, **asmby**, or **always** is invalid. If you use any one or all of **string**, **asmby**, or **always** then the use of **all** is invalid. The following are some typical syntax examples of using **FTERM**.

FTERM/PART[,all][,IFERR,label:]

FTERM/PART[,string] [asmby] [,always] [,IFERR,label:]

FTERM/TXT,file#[,IFERR,label:]

Parameters **PART**
 Minor word that indicates that the files to be closed are part files.

[options]
 Specifies either **all** or any of the options **string**, **asmby**, or **always**.

all
 Closes all parts in a session without saving. When you use the **all** option, the **string**, **asmby**, and **always** options are not available for use.

string
 Specifies the name of the part to close. The part name can be a file specification (path and file name). The string can be a variable or a string literal.

asmby
 Closes the current part/specified part and all of its subassemblies.

always
 Closes the part(s) without error even if the part(s) were modified.



TXT

Minor word that indicates that the file to be closed is the text file which is currently in the specified scratch file.

file#

The number of the scratch file which is to be closed. Ten scratch file areas are available, therefore, the specified number must be between 1 and 10.

IFERR,label:

Specifies a label to which the program jumps if an error occurs.

<i>Name</i>	FDEL	Delete a File
<i>Synopsis</i>	FDEL/'filespec'[,IFERR,label:]	
<i>Description</i>	Deletes the specified file if the user has proper access to the file.	
<i>Parameters</i>	'filespec'	The filespec is the pathname plus the filename you want. If the file you need is in your current directory, the filename should be sufficient to find it. The FDEL statement requires that the file extension (e.g., .PRT, .GRS) be specified.
	IFERR,label:	Specifies a label to which the program jumps if an error occurs.

Object Cycling

There are four GRIP commands to examine objects in the data model or to perform some function on each object. Two commands cover geometric objects; the remaining two commands are used for non-geometric objects.

Table 9–2 Data Cycling Statements

Function	Format
Initialize Data Model for object cycling	INEXTE[/ALL]
Obtain the next selectable object	NEXTE/IFEND,label:
Initialize Data Model for non geometric cycling	INEXTN/{type no. type GPA} [,subtype][,IFERR,label:]
Obtain the next non–geometric object	NEXTN/IFEND,label1: [,IFERR,label2:]

You must use the INEXTE or INEXTN commands to start the cycle at the beginning of the data model. To access selectable geometric objects, use the NEXTE command in a loop. To access non–geometric objects, use the NEXTN command in a loop. You should not delete objects in a loop. The NEXTE and NEXTN commands use data from the 'current' object to find the next object. If you delete the object, the subsequent NEXTE or NEXTN command will fail.

Example 1–2.

You may only access objects on visible and selectable layers. For example, think of the objects as being in a list as shown in the figure below. The list shows a part with 9 objects:

- 4 on Visible and Selectable layers (L1, L2, C1, P1)
- 3 on Invisible and Non-selectable layers (L3, C2, P2)
- 2 on Visible and Non-selectable layers (E(1..2))

Visible Selectable	Invisible Non-selectable	Visible Non-selectable
L1		
L2		
	L3	
C1		
	C2	
		E(1)
		E(2)
P1		
	P2	
	(LAST OBJECT)	

Figure 9–1 Visible Selectable Layers



INEXTE: Initialize Data Model Cycling

Synopsis

INEXTE[/ALL]

Description

This statement initializes or resets the data model cycle mechanism so that the next use of the **NEXTE** statement will access the first selectable, unblanked object in the data model which is inside the clipping bounds.

Parameters

/ALL

Minor word that indicates that all of the selectable objects in the data base including objects blanked and outside the clipping bounds will be accessed. If **/ALL** is not used, only the currently displayed objects will be accessed.

NEXTE: Cycle to Next Object*Synopsis***obj = NEXTE/IFEND,label:***Description*

The following is an entity valued function which may be used to address the next selectable object in the data model.

*Parameters***IFEND**

Minor word that indicates that the next parameter will be a statement label, which the program will branch to after the last selectable object in the data model has been accessed.

label:

The statement label to which the program will branch after the last selectable object in the data model has been accessed.

Example

This code changes the color of all lines on selectable layers to red. The **ALL** minor word in the **INEXTE** statement causes all objects in the data model to be examined, even those that are blanked or outside the clipping bounds (not displayed). The **MASK** statement causes only lines in the database to be selectable.

```
ENTITY/ cur_obj
```

```
    INEXTE/ ALL
    MASK/ 3
```

```
WHL010:
```

```
    cur_obj = NEXTE/ IFEND, WHL020:
    &COLOR(cur_obj) = &RED
    JUMP/ WHL010:
```

```
WHL020:
```

```
    HALT
```

INEXTN: Initialize Non–geometric Object Cycling

Synopsis

INEXTN/{type no. | type GPA} [,subtype] [,IFERR,label:]

Description

This statement initializes or resets the data model cycle function so that the next use of the **NEXTN** statement will access the first non-geometric object. **INEXTN** and **NEXTN** are used to cycle only non-geometric objects.

Parameters

type no.

A numerical value or variable which specifies the type of object that can be accessed using the **NEXTN** statement. Allowable values are:

Value	Object Type
12	Category
60	View
61	Layout
62	Drawing
64	Reference set/library entry
100	Machining operation
107	Machining parameter set
109	Machining tool

type GPA

A GPA which specifies the type of object that can be accessed using the **NEXTN** statement. Allowable GPAs are:

GPA	Object Type
&VIEWS	View
&LAYOUT	Layout
&DRWG	Drawing
&REFLIB	Reference set/library entry
&OPER	Machining operation
&PARSET	Machining parameter set
&TOOLS	Machining tool
&CATGRY	Category

subtype

A numerical value or variable which specifies the subtype of object that can be accessed using the **NEXTN** statement. If the subtype is not specified, then all subtypes of the given type will be returned. Allowable subtypes are shown in the table below.

Type /Desc	Subtype	Subtype Description
64=Reference set/library entry	0	Reference set
	1	Tool from library entry
	2	Parameter set from library entry
107=Machining parameter set	11	Mill post commands
	13	Lathe post commands
	110	Pocket
	210	Fixed axis surface contouring
	211	Variable axis surface contouring
	220	Interactive generated sequential surface machining
	222	GRIP generated sequential surface machining
	230	Parameter line machining
	240	Zig-zag surface
	250	Rough-to-depth
	260	Cavity milling
	310	Lathe rough
	320	Lathe finish
	330	Lathe groove
	340	Lathe thread
	350	Lathe drill
	450	Point-to-point
460	Sequential curve mill	
461	Sequential curve lathe	

IFERR,label:

Specifies a label to which program execution jumps if an error occurs.



NEXTN: Cycle to Next Non–geometric Object

Synopsis

str = NEXTN/IFEND,label1:[,IFERR,label2:]

Description

This statement will return the name of the next non–geometric object in the data base. This statement is used to cycle only non–geometric objects and must be used with **INEXTN**. One name at a time is returned for each **NEXTN** statement issued.

Parameters

IFEND,label1:

Specifies a label to which program execution will jump after the last non–geometric object in the part file has been accessed.

IFERR,label2:

Specifies a label to which program execution will jump if an error occurs.

Non-Geometric Database Cycling Example

Example

This code changes sends a list of drawings in the current part file to the listing window.

A while loop is simulated using labels and a jump statement. This loop will execute as many times as needed to find all of the lines in the database.

```
STRING/ dwg_name(30)

      PRINT/ 'List of Drawings'
      PRINT/ '-----'
      PRINT/ ' '
      INEXTN/ &DRWG, IFERR, ERR100:

WHL010:
      dwg_name = NEXTN/ IFEND, WHL020:,$
                        IFERR, ERR110:
      PRINT/ dwg_name
      JUMP/ WHL010:
WHL020:
      JUMP/ CANCEL:

ERR100:
      MESSG/ 'No Drawings Found'
      JUMP/ CANCEL:
ERR110:
      MESSG/ 'Error Reading Drawing Name'

CANCEL:
      HALT
```

Reading and Writing to Text Files

The commands available to open, read from, and write to text files are described in the following pages.

Table 9–3 File Access Commands

Function	Format
File Pointer Control	RESET/file#
Read Text	READ/file#[,LINNO,line#][,USING,'image string'[,IFEND,label:][,IFERR,label:],variable list
Write Text	WRITE/file#[LINNO,line#][,USING,'image string,],data list
Print to a file or listing window	PRINT/[USING,'image string',]data list

<i>Name</i>	RESET	File Pointer Control
<i>Synopsis</i>	RESET/file#	
<i>Description</i>	This statement sets the line pointer to zero in the specified scratch file.	
<i>Parameters</i>	file#	The number of the scratch file in which the line pointer is to be positioned. Ten scratch file areas are available, therefore, the specified number must be between 1 and 10.

READ: Read Text from a File

<i>Name</i>	READ	Read Text
<i>Synopsis</i>	READ/file#[,LINNO,line#][,USING,'image string'] [,IFEND,label:][,IFERR,label:],variable list	
<i>Description</i>	The following statement reads a line of data from the specified scratch file. An unformatted READ of a string will discard leading blanks. A formatted READ of a string will retain leading blanks. Unless the program changes line numbers using the RESEQ statement, all line numbers are in increments of 10.	
<i>Parameters</i>	<p>file# The number of the scratch file from which the data is to be read. Ten scratch file areas are available, therefore, the specified number must be between 1 and 10.</p> <p>LINNO Minor word which indicates that a specified line is to be read.</p> <p>NOTE If LINNO is not specified the line following the line pointer position in the data file will be read and the line pointer will be set to the last line read.</p> <p>line# The number of the line to be read.</p> <p>USING Minor word which indicates that the data is to be input to a specified format.</p> <p>NOTE The USING option should be used only when the data in the file is in a free format.</p> <p>'image string' The format image string is covered in the GRIP Programming Reference Manual.</p> <p>IFEND,label: Specifies a label to which the program jumps when the pointer reaches the end of the file.</p>	

IFERR,label:

Specifies a label to which the program jumps if an error occurs.

variable list

The variable list may consist of any combination of string variables or numerical variables.

NOTE

If the number of items in the data record exceed the number of variables in the variable list, the remaining items will be ignored. If the number of variables in the variable list exceed the number of items in the data record, subsequent lines of data will read until the variable list has been satisfied.

The variable list may contain a maximum of 42 variables and/or array positions. However, the line length is limited to 132 characters. If this length is exceeded, the remaining characters are ignored.



WRITE: Write Text to a File

<i>Name</i>	WRITE	Write Text
<i>Synopsis</i>	WRITE/file#[,LINNO,line#][,USING,'image string'],data list	
<i>Description</i>	This statement writes a line of data to the specified scratch file.	
<i>Parameters</i>	<p>file# The number of the scratch file to which the data is to be written. Ten scratch file areas are available, therefore, the specified number must be between 1 and 10.</p> <p>LINNO Minor word which indicates that the line created will have the specified line number. If the specified line exists, the data will be replaced.</p> <p>NOTE If LINNO is not specified in the first WRITE statement, the beginning line number will be 10. If LINNO is not specified on subsequent WRITE statements, the new line number will be equal to the last line number plus 10.</p> <p>line# The number of the line to be created.</p> <p>USING Minor word which indicates that the data is to be output to a specified format.</p> <p>'image string' The format image string which is covered in the GRIP Programming Reference Manual.</p> <p>data list The data list may consist of any combination of string variables, numerical variables, string literals or numerical values.</p>	

PRINT: Print Data on Listing Device

<i>Synopsis</i>	PRINT/[USING,'image string',]data list
<i>Description</i>	This statement lists the specified data on the current listing device.
<i>Parameters</i>	USING Minor word which indicates that the data is to be output to a specified format. 'image string' The format image string is covered in the GRIP Programming Reference Manual. data list The data list may consist of any combination of string variables, numerical variables, string literals or numerical values.
<i>Description</i>	The PRINT statement is a convenient way of displaying data to the user. Possible uses include help messages, tables of information, and debugging.

Activity: Reading a Text File

Modify the L–Shape program to read in the dimensions from a text file based on the selection of a configuration from a menu.

- ❑ Replace the PARAM/ statement with the following CHOOSE/ statement

```
CHOOSE/ 'Select Configuration' , $
        'LSHAPE 1' , $
        'LSHAPE 2' , $
        'LSHAPE 3' , $
        'LSHAPE 4' , $
        'LSHAPE 5' , resp
```

- ❑ Fetch the text file **grp_lshape.dat** and read data from each line until the correct “dash” number is found.

```
READ/2, IFEND, CANCEL:.,dsh,bw,tw,tht,tpht
```

The text file contains the following lines of data:

```
1, .25, 1, 2, .5
2, .5, 1.5, 4, 1
3, 1, 2, 6, 2
4, 1.5, 3, 8, 3
5, 2, 5, 10, 5
```

(This Page Intentionally Left Blank)



Object Access

Lesson 10

Objectives

- Understand Layer Control.
- Understand Layer Categories.
- Control object selection in a GRIP program.
- Obtain object information.
- Use Global Parameter Access Symbols to control the Unigraphics environment.



Categories and Layer Control

The GRIP statements used to control layers perform the same functions which are defined in Unigraphics using Layer Setting dialogs. The **LAYER** command controls which layers are selectable in a Unigraphics part.

Below is the GRIP layer control statement which will be described in this manual, as well as the commands to access objects and remove, hide, or delete objects in a partfile.

Table 10–1 Layer Control and Modification Command

Function	Format
Layer Control	LAYER/{[WORK,n], [ACTIVE,{REST layer list [,CAT,'cat']}] [REF,{REST layer list[,CAT,'cat']}] [INACT,{REST layer list[,CAT,'cat']}]}
Create Category	CAT/'name' {,layer list ,CAT,'cat'} [,DESCR,'description'][,IFERR,label:]
Edit Category	CATE/'name' {,ADD ,REMOVE} [layer list][,CAT,'cat list'] [,DESCR,'description'][,IFERR,label:]
Delete Category	CATD/'name'[,IFERR,label:]
Query Category	CATV/'name'[,LAYER,layers,CNT,count] [,DESCR,'description'][,IFERR,label:]

LAYER: Layer Control

Synopsis

LAYER/[**WORK**,*n*],
 [**ACTIVE**,{**REST**|layer list[,**CAT**,'cat'']}],
 [**REF**,{**REST**|layer list[,**CAT**,'cat'']}],
 [**INACT**,{**REST**|layer list[,**CAT**,'cat'']}]

Description

Establishes the operating status of the 256 available layers. You can use the EDA **&LAYER** to later change the layer of a specified object.

Parameters

WORK

Minor word which indicates that the following parameter will establish the work layer. The work layer is the only layer on which objects may be created.

n

The number of the work layer.

ACTIVE

Minor word which indicates that the following minor word or list will establish the active layers. Objects on active layers are visible and selectable.

REF

Minor word which indicates that the following minor word or list will establish the reference layers. Objects on reference layers are visible, but not selectable.

INACT

Minor word which indicates that the following minor word or list will establish the inactive layers. Objects on inactive layers are neither visible nor selectable.

REST

Minor word which follows each of the minor words **ACTIVE**, **REF** and **INACT** and may appear only once in a layer statement. **REST** indicates that the status of all layers not previously determined in the statement will assume the status of the preceding minor word.

layer list

Each of the minor words **ACTIVE**, **REF**, and **INACT** may be followed by an optional list of layer numbers separated by commas which will assume the status of the associated minor word. The layer list may contain either a list of layer values separated by commas and/or a subrange of layers (1..256).

CAT

Minor word which indicates that an existing category name will be specified.

'cat'

A string or string variable which represents a previously defined layer category name which, if used, must be last in the layer list.

'cat'

A string or string variable which represents a previously defined layer category name. This string cannot contain any blanks.

CAT: Create Layer Category

Synopsis

CAT/'name'[,layer list][,CAT,'cat'][,DESCR,'description']
[,IFERR,label:]

Description

Allows specified layers to be grouped together and given a name which may subsequently be used in the layer statement. You can use categories to manage ranges or sets of layers at one time. Category members may overlap, and a layer may be a member of an unlimited number of categories.

For example, you may be working on an assembly of a product, detailing one of the parts. You could create a category and give it the name of the part. The category would contain the layers which hold the part dimensions, notes and title block, geometry, etc. You could then automatically activate the entire part by activating that category.

You can also create a description for the category by using the minor word **DESCR** followed by the description string. The description string can be a maximum of 80 characters long.

If you specify an existing category name, this command completely redefines the category.

NOTE

The layer list and minor word **CAT** are both optional parameters. However, if neither is specified, the system doesn't generate an error message *and* it doesn't create any categories. You should specify either one or both of these optional parameters to create a category.

Parameters

'name'

The category name which may be up to 30 characters long and must start with an alpha character.

layer list

A list of layer numbers separated by commas and/or a subrange of layers.

CAT

Minor word indicating that an existing category name will be specified.



'cat'

A string or string variable which represents a previously defined layer category name. This string cannot contain any blanks.

DESCR

Minor word indicating that a description will be specified for the category.

'description'

A description string of up to 80 characters.

IFERR,label:

Specifies a label to which program execution jumps if an error occurs.

Example

This example demonstrates the creation of a category containing several layers and a description.

Create Category

CAT/'DIMS', 10..20, DESCR,'Dimensions, Symbols, and Notes'

The layers containing the dimensions, drafting symbols and notes (layers 10 through 20) are now grouped into the category called DIMS. A description of the category's contents accompanies DIMS. At this point, if the complete assembly was active (layers 1-256), you could look at just the dimensions by making all layers (except the work layer) INACTIVE, then ACTIVATING the category DIMS.

LAYER/WORK,10,ACTIVE,CAT,'DIMS',INACT,REST



CATE: Edit Category

<i>Synopsis</i>	<p>CATE/'name' {,ADD ,REMOVE} [,layer list][,CAT,'cat list'][,DESCR,'description'] [,IFERR,label:]</p>
<i>Description</i>	<p>Allows you to edit existing categories. To edit a category, you must specify one of the minor words, ADD or REMOVE. Both of these minor words apply to the layer list, category list, and description field. When the minor word DESCR is present, the description string is required when adding, but is optional and is ignored when removing.</p>
<i>Parameters</i>	<p>'name' The name of the existing category to edit.</p> <p>ADD Minor word which enables you to add layers or to add or change the description field.</p> <p>REMOVE Minor word which enables you to remove layers or the description field.</p> <p>layer list A list of layer numbers separated by commas and/or a subrange of layers. To specify a subrange of layers, use two periods to specify the range. For example, use 11..20 to specify layers 11–20.</p> <p>CAT Minor word specifies the use of an existing category name.</p> <p>'cat list' A string list which represents previously defined layer category names. The strings cannot contain any blanks.</p> <p>DESCR Minor word indicates that there is a description for the category.</p> <p>'description' A description string of up to 80 characters.</p> <p>IFERR,label: Specifies a label to which program execution jumps if an error occurs.</p>

CATD: Delete Category

Name **CATD** Delete Category

Synopsis **CATD/'name'[,IFERR,label:]**

Description Allows you to delete a category.

Parameters 'name'
The name of the existing category to delete.

IFERR,label:
Specifies a label to which program execution jumps if an error occurs.

CATV: Query Category

<i>Synopsis</i>	CATV/’name’[,LAYER,layers,CNT,count][,DESCR,’description’] [,IFERR,label:]
<i>Description</i>	Allows you to query the layers and the description in a category.
<i>Parameters</i>	<p>’name’ The name of the existing category to query.</p> <p>LAYER Minor word which indicates that you want to verify the layers.</p> <p>layers The layers in the category are returned in this numeric array. This array should be large enough to hold the layers in the category.</p> <p>CNT Minor word which indicates the following number variable is the return area for the number of layers in the category.</p> <p>count Numerical return value which holds the actual count of layers in the category.</p> <p>DESCR Minor word indicates that you want to verify the description.</p> <p>’description’ A return area for the description string which can consist of up to 80 characters.</p> <p>IFERR,label: Specifies a label to which program execution jumps if an error occurs.</p>

Object Control

In Unigraphics, the selection of objects can be controlled by the user using the Unigraphics Class Selection Menu, blanking status, layer status, and group status. In a GRIP program, various methods of specifying and selecting objects are available.

- Many GRIP commands accept an **entity list** as an argument to the command. An **entity list** is a list of object variable names. These variables can reference simple objects, object arrays, and groups of objects. NO TAG shows a variety of entity lists used with the DELETE statement.

Example

```
ENTITY/ refpt,pt(100),gp, ln(60)
.
.
.
DELETE/ refpt, gp
DELETE/ pt(2), pt(6), pt(8)
DELETE/ pt(30..60), ln
```

- The **MASK** statement controls the selection of objects according to their type. For instance, only circles can be selectable as in NO TAG

Example

```
MASK/ 5
```


The GRIP Statements used to temporarily hide objects from the display and delete objects from the database are included in this section. The statement used to obtain information about objects in the database and to filter on the types of objects to be selected are also covered.

Table 10–2 Layer Control and Modification Command

Function	Format
Mask objects(s)	MASK/{ALL NONE [OMIT,],obj type list}
TYPF of object	num = TYPF(<u>obj</u>)
Obtain object information	OBTAIN/ <u>obj</u> ,variable list
Draw Screen	DRAW/{ON OFF ALL <u>obj list</u> }
Delete object(s)	DELETE/{ <u>obj list</u> ALL}

MASK: Class Selection*Synopsis***MASK/{ALL|NONE|[OMIT],obj type list}***Description*

Changes the selectable status associated with a particular object type to valid or invalid.

The **MASK** statement can be used to make only certain objects selectable by their object type. For example, you may want to delete all objects except lines. No matter what the lines are named (even if they are not named) they can be made non-selectable by their object type.

```
ENTITY/PT1,CR(2),L(3),LN1,L1
PT1=POINT/0,0
CR(1)=CIRCLE/1,1,.5
CR(2)=CIRCLE/2,2,1
LN1=LINE/0,0,1,1
L(1)=LINE/2,2,3,3
LINE/3,4,1,6
```

```
MASK/OMIT,3
DELETE/ALL
```

*Parameters***ALL**

Minor word that indicates that all of the object types in the data model are valid and selectable.

NONE

Minor word that indicates that all of the object types in the data model are invalid and not selectable.

OMIT

Minor word that indicates that all of the following object types in the data model are invalid and not selectable. If the minor word **OMIT** is not used, only those object types included in the list will be valid and selectable.

obj type list

A list of object types. See the following table for allowed object type values.

Object Type	Value
Point	2
Line	3
Circle	5
Conics	6
B-curve	9
Pattern	10
Boundary	14
Groups	15
Cylinder Face	16
Cone Face	17
Sphere Face	18
Revolved Face	19
Extruded Face	20
Planar Face	22
Blend Face	23
Drafting Object	25
Dimension	26
Margin	29
B–Surface face	43
Coordinate System	45
Plane	46
Component	63
Offset surface face	65
Foreign surface face	66
Solid/sheet body	70
Face	71
Edge	72
Section Edge	199
Silhouette	201
Section Line	202

NOTE Types 16–20, 22, 43, 65, and 66 are special cases of a face (type 71). Cylindrical, conical, etc. solid and sheet bodies correspond to type 70. For example, type 16 corresponds to the cylindrical face of a solid cylinder or the face of a cylindrical sheet body. It is not the solid or sheet cylinder: That object is a type 70.

TYPF: Object Type

<i>Synopsis</i>	num = TYPF(obj)
<i>Description</i>	Returns an integer which represents the type of object in the argument.
<i>Parameters</i>	<p>obj An existing object.</p> <p>For a complete list of all Unigraphics object types, see Appendix C.</p>
<i>Example</i>	This example demonstrates the use of the TYPF statement to extract the object type number from several objects.
<i>Declarations</i>	ENTITY/PT1, LN1, DIM1 STRING/STR1(20)
<i>Geometry Definition</i>	PT1=POINT/0,0 LN1=LINE/0,0,1,1 DIM1=LDIM/HORZ,,.5,2,PT1,XLARGE, LN1 STR1='THE OBJECT NUMBER='
<i>Object type</i>	NUMPT1=TYPF(PT1) NUMLN1=TYPF(LN1) NUMDIM=TYPF(DIM1)

The numerical variables would be assigned values as follows:

Variable	Numerical Value Extracted
NUMPT1	2
NUMLN1	3
NUMDIM	26

You might want to continue your program to send the value to your message monitor as follows:

```
MESSG/STR1+ISTR(NUMPT1)
MESSG/STR1+ISTR(NUMLN1)
MESSG/STR1+ISTR(DIMNUM)
```

You could also write a GRIP program to cycle through your data base and find all dimensions (26) and delete them.

The type EDAs are listed below:

Function	EDA Symbol	Access Type	Data Type
Object Type	&TYPE(<u>obj</u>)	RO	Number
Object Subtype	&SUBTYP(<u>obj</u>)	RO	Number

Some objects have a type and a subtype. The **&TYPE** and **&SUBTYP** EDAs can be used in conjunction with each other as follows:

```
ENTITY/ obj
NUMBER/objtyp, objsub

objtyp = &TYPE(obj)
objsub = &SUBTYP(obj)
IFTHEN/ objtyp == 25
  IFTHEN/ objsub == 1
    MESSG/ 'Object is a Note'
  ELSEIF/ objsub == 2
    MESSG/ 'Object is a Label'
  ENDIF
ENDIF
HALT
```

OBTAIN: Object Information

Synopsis

OBTAIN/obj,variable list

Description

Causes the numerical data stored with an object to be assigned to numerical variables.

Parameters

obj

The existing object from which the variables in the variable list are set.

variable list

The variables which are assigned the parameter values obtained from the object. The number of variables required and/or their position in the list is dependent on the object type. Although the number of variables in the list and the number of parameters obtained do not have to be the same, the sequence of the variables in the list is extremely important. Any unmatched variables or parameters are ignored.

The following table consists of the various object types, the parameters which may be obtained, and their relative positions.

See the GRIP Programming Manual for information concerning items highlighted by single, double, and triple asterisks (*).

Object Type	Object Type #	Parameters and Definitions
Point	2	OBTAIN/ <u>obj</u> ,X,Y,Z <u>obj</u> -- point object name X -- coordinate Y -- coordinate Z -- coordinate
Line	3	OBTAIN/ <u>obj</u> ,X1,Y1,Z1,X2,Y2,Z2 <u>obj</u> -- line object name X1 -- start point X coordinate Y1 -- start point Y coordinate Z1 -- start point Z coordinate X2 -- end point X coordinate Y2 -- end point Y coordinate Z2 -- end point Z coordinate

Object Type	Object Type #	Parameters and Definitions
Circle	5	<p>OBTAIN/obj,X,Y,Z,R,A1,A2</p> <p>obj -- circle object name X -- center point coordinate Y -- center point coordinate Z -- center point coordinate R -- radius A1 -- start angle of arc A2 -- end angle of arc</p>
Ellipse	6	<p>OBTAIN/obj,X,Y,Z,S1,S2,ROT,A1,A2</p> <p>obj -- ellipse object name X -- center point coordinate Y -- center point coordinate Z -- center point coordinate S1 -- semi-major axis length S2 -- semi-minor axis length ROT -- rotation angle A1 -- start angle of conic A2 -- end angle of conic</p>
Hyperbola	6	<p>OBTAIN/obj,X,Y,Z,S1,S2,ROT,YMIN, YMAX</p> <p>obj -- hyperbola object name X -- center point coordinate Y -- center point coordinate Z -- center point coordinate S1 -- semi-transverse axis length S2 -- semi-conjugate axis length ROT -- rotation angle YMIN-- minimum Y distance YMAX-- maximum Y distance</p>
Parabola	6	<p>OBTAIN/obj,X,Y,Z,F,ROT</p> <p>obj -- parabola object name X -- vertex coordinate Y -- vertex coordinate Z -- vertex coordinate F -- focal length ROT -- rotation angle</p>



Object Display Control

When you use a statement to create an object such as:

$$LN = LINE/1, 1, 2.5, 3.7$$

the object is added both to the data model, and to a list of objects (called the display list or display buffer) to be displayed. However, the object is not displayed on the screen.

In order for you to see the objects in the buffer displayed on the screen, one of these conditions must occur:

- Buffer becomes full (normally 511 objects).
- Execution reaches an interactive command.
- A DRAW/ALL or DRAW/obj statement is reached
- Execution reaches a HALT statement.
- A run-time error occurs.

DRAW: Control Object Display

Synopsis **DRAW/{ON|OFF|ALL|obj list}**

Description Controls the display of objects. When an object is defined, it is added to the database and also to the display list. The display list which may contain up to 511 objects is periodically emptied and the objects displayed. This display takes place under one of the following occurrences:

- Before any interactive command is executed.
- When the **HALT** function is executed.
- When the display list is full.
- When a run time error occurs.
- When you issue the DRAW/ALL command.

Parameters **ON**
Minor word that causes normal display operations to occur, including adding subsequently defined objects to the display list. DRAW/ON is the default.

OFF
Minor word that suspends all display operations. This includes not adding subsequently defined objects to the display list, as well as not changing the display for commands such as VIEWE (Edit View) or LAYR(Retrieve Layout). If you want to see the result of these while DRAW/OFF is in effect, use DRAW/ALL, which ignores the DRAW/OFF setting. While in effect, DRAW/OFF also suppresses temporary display such as asterisks and arrows.

ALL
Minor word that regenerates the display of all objects including those which were previously inhibited (either interactively or with the DRAW/OFF command). The display list is emptied and rebuilt to include the regenerated objects. This minor word does not change the display state as specified by DRAW/ON or DRAW/OFF.

obj list
Causes all objects in the object list, which may have been inhibited, to be added to the display list. If the object is a group, it is not displayed. If you wish to display group members, you must put each in a separate statement.



Example This example demonstrates the use of the **DRAW** statement.

Declarations ENTITY/LN1, LN2, LN3, LN4, CR1, CR2

Geometry Definition LN1=LINE/-2,0,0,-1
LN2=LINE/0,-1,0,1
LN3=LINE/0,1,-2,0

Draw Statement DRAW/OFF
CR1=CIRCLE/-.625,0,.5
CR2=CIRCLE/.625,0,.5

Draw Statement DRAW/ON
LN4=LINE/-2,0,0,0
HALT

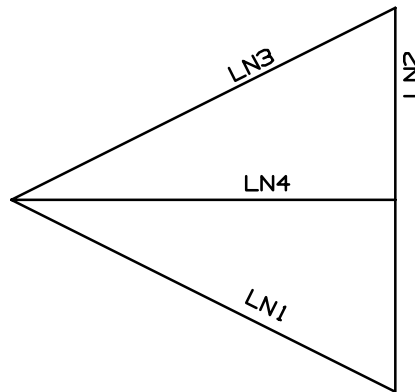


Figure 10–1 Use of the DRAW/OFF and DRAW/ON statements

NOTE If certain system and drafting parameters such as the line font, dash size, and character size have been changed, the objects in the display list will assume those characteristics.

Object Access

Example This example demonstrates the use of the **DRAW** statement.

Declarations ENTITY/LN1, LN2, LN3, LN4, CR1, CR2

Geometry Definition LN1 = LINE/-2,0,0,-1
LN2 = LINE/0,-1,0,1
LN3 = LINE/0,1,-2,0

Draw Statement DRAW/OFF
CR1 = CIRCLE/-.625,0,.5
CR2 = CIRCLE/.625,0,.5

Draw Statement DRAW/ON
LN4 = LINE/-2,0,0,0

Draw Statement DRAW/CR1, CR2
HALT

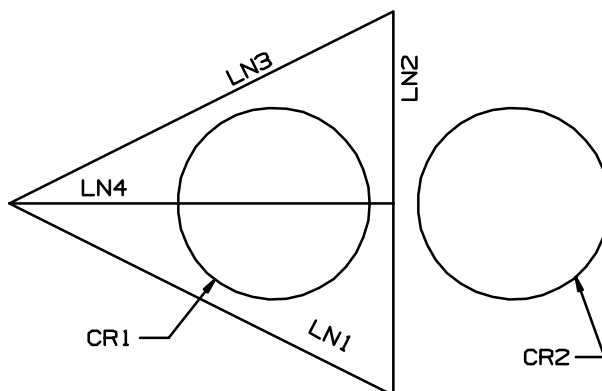


Figure 10–2 Use of the DRAW/OFF, DRAW/ON and DRAW/obj statements

DELETE: Delete

<i>Synopsis</i>	DELETE /{ <u>obj list</u> ALL }
<i>Description</i>	Permanently removes objects from the viewing screen and the data model.
	NOTE To remove a component, use the DELETE command.
	NOTE Sectioning objects, including section lines and system generated crosshatching, cannot be deleted using the DELETE command. Use the VIEWD command to remove the section view. If a detail of a section view is removed, the section line remains intact until removal of the parent section view.
<i>Parameters</i>	<p>obj list A list of existing objects which are to be deleted.</p> <p>ALL Minor word which indicates that all objects which satisfy the following conditions are to be deleted:</p> <ol style="list-style-type: none"> 1. must be a selectable object type/subtype 2. must not be blanked 3. must reside on a layer which is selectable in the work view
<i>Example</i>	This example demonstrates the use of the DELETE statement.
<i>Declarations</i>	ENTITY/P(3),LN1,CR1
<i>Geometry Definition</i>	P(1)=POINT/0,-.5 P(2)=POINT/0,0 P(3)=POINT/1,1.5 LN1=LINE/-1,.5,1,.5 CR1=CIRCLE/CENTER,P(1),TANTO,LN1
<i>Objects Deleted</i>	DELETE/LN1,P
	NOTE Objects which may be necessary for the construction of other objects, such as the line LN1, may be deleted when they are no longer of any value.
	All of the points in the array P are deleted.

Global Parameter Access (GPA) Symbols

Global Parameter Access (GPA) symbols allow you to access the global preferences of a part while in a GRIP program. The global preferences you have available include many of the parameters found in the Modeling and Drafting Preference and WCS pulldown menus. The first character of a GPA is always an ampersand (&) followed by from 1 to 6 alpha characters.

Each GPA symbol is associated with either a unique global parameter, or a constant, and has an access type, a data type, and a data range. Access types, data types and ranges are described below.

Access Type (Read and Write Options) of GPA Functions

Access type defines the read/write status of the GPA. GPAs contain data which can be extracted (read) for use in your program. Some symbols allow you to directly alter the data (write) by assigning the properly valued data to the GPA. Abbreviations of access types used in the GPA format summary are as follows:

- RO (read only)
- WO (write only)
- RW (read write)
- C (constant) read only

Data Types Assigned to GPAs

Data type defines whether a particular GPA is entity, number, or string valued. The data associated with a GPA, if entity valued, must be treated like an object as it is assigned or extracted. If you read an entity valued GPA and assign the data to a variable, the variable must be declared as an ENTITY.

The data types for GPA symbols are the following:

- NUMERICAL (N)
- ENTITY (E)
- STRING (S)
- NUMERICAL ARRAY (NA(I))
- OF DIMENSION(I)



Entity Valued GPAs

Entity valued GPAs return a specific object from the work part. Variables which store these values must be declared with the ENTITY/ statement.

Example Part attribute entity valued GPA

```
ENTITY/PART
PART=&PARATT
```

Number Valued GPAs

Number valued GPAs return a number within a specified range. Variables which store these values should be declared with the NUMBER/ statement.

Example Part units number valued GPA

```
NUMBER/ unts
unts = &UNIT
IFTHEN/ unts == 1
    val=val/25.4
ENDIF
```

String Valued GPAs

String valued GPAs, when assigned to a variable, must be declared as a STRING as illustrated in the example below.

Example Work view string valued GPA

```
STRING/cur_view(30)
cur_view = &WORKVW
.
```

GPA Range

The data range represents the set of values the global parameter may assume. Range defines the value which can be assigned to a GPA. Some GPAs have multiple parameters (e.g. &ON or &OFF) which have numerical values that lie within their range. These parameter options have a constant, single numerical value which cannot be changed (e.g. the value of &ON is always 1). The range of string valued GPAs is defined in a number of characters allowed in the string (e.g. &DEFCLS has a range of 8 characters).

System Constants

The GPAs that deal with module preferences are not associated with any other GPAs and have no data type, access type, or range. These GPAs are as follows:

```
&NULENT
&NULSTR
&PI
```

&NULENT determines if an object identifier exists in the data base by testing its status. LN1 could be tested as follows:

Example Using the &NULENT GPA constant

```
IF /LN1 <> &NULENT , DELETE /LN1
```

If LN1 is not a null object, it will be deleted. This can be used to avoid run time errors when the **DELETE** statement is used with a null object.

&NULSTR works like &NULENT except that it tests to see if a string has no characters. For example, you could test to see if STR contained any characters as follows:

Example Using the &NULSTR GPA constant

```
IF /STR == &NULSTR , JUMP /LBL010 :
.
.
.
LBL010 : TEXT / 'NO CURRENT TXT-ENT TEXT' , STR , RESP
JUMP /LBL010 : , TERM : , , , , RESP
```

If STR was a null string, the user will be prompted to enter the text.

&PI contains the constant value for PI. It can be used anywhere a numerical value is valid.

Example The &PI GPA constant

```
ENTITY/PT1 , PT2 , CR1
PT1=POINT/0 , 0
PT2=POINT/1 , 1.5
CR1=CIRCLE/CENTER , PT1 , PT2
R=&RADIUS ( CR1 )
ANS=&PI * ( R**2 )
```

ANS would equal the area of CR1.

Active Part Status

Function	GPA Symbol	Access Type	Data Type
Active Part Status	&ACTPRT	R/O	Number [1..2]

For the **&ACTPRT** GPA, the following values are returned:

1 = No Active Part

2 = Active Part

In programs that create or access part file information, a check should typically be made to insure that a part file is active. This can be done using the **&ACTPRT** GPA, as follows

Example

```
IFTHEN/ &ACTPRT == 1
  MESSG/ 'Active part not available', $
        'Open file and restart program'
  JUMP/ CANCEL:
ENDIF
```

Decimal Places

Function	GPA Symbol	Access Type	Data Type
Places Past Decimal Point	&DECPL	R/W	Number [0..9]

Controls the places displayed to the right of the decimal for numerical displays. This GPA affects the decimal places displayed for numerical displays, such as the **PRINT** command. **&DECPL** also affects the **FSTR** and **FSTR** functions discussed in Chapter 7.

Example

```
NUMBER/sv_decpl

sv_decpl = &DECPL  $$ save original
&DECPL = 2
PRINT/ 'Current hole radius = '+FSTR(hr)
&DECPL = sv_decpl  $$ restore original
```

Work Layer

Function	GPA Symbol	Access Type	Data Type
Current Work Layer	&WLAYER	R/W	Number [1..256]

Controls or returns the number of the current work layer.

Example

```
NUMBER/ sv_wlayer

sv_wlayer = &WLAYER      $$ save original
&WLAYER = 50
hol_arc = CIRCLE/ xc, yc, zc, hr
&WLAYER = sv_wlayer     $$ restore original
```

Work View

Function	GPA Symbol	Access Type	Data Type
Current Work View	&WORKVW	R/W	String 30 characters

Changes or returns the current work view. When changing the work view, the desired new work view must be a part of the current layout.

Example

```
STRING/ sv_workvw(30)

sv_workvw = &WORKVW     $$ save original
&WORKVW = 'RIGHT'
.
.
&WORKVW = sv_workvw    $$ restore original
```

Unit of Measurement

Function	GPA Symbol	Access Type	Data Type
Examine Unit of Measurement	&UNIT	R/O	Number [1..2]

For the **&UNIT** GPA, the following values are returned:

- 1 = Inches**
- 2 = Millimeters**

Example

```
IFTHEN/ &UNIT == 1
  MESSG/ 'Part is English'
ELSE/
  MESSG/ 'Part is Metric'
ENDIF
```

User ID

Function	GPA Symbol	Access Type	Data Type
User Identification String	&USERID	RO	String 30 Char

Returns the current user ID to a receiving variable or GRIP statement.

Activity: Entity Cycling

Write a GRIP program to cycle through selectable objects in a part file, move the objects to different layers based on their type, and create layer categories.

- Open the part **grp_cycle.prt**.
- Create the following layer categories:

<u>Category Name</u>	<u>Layer</u>
MODEL	1–100
SOLIDS	1–20
CURVES	41–60
DATUMS	61–80

- Use the INEXTE and NEXTE commands to cycle through selectable objects.
- Move the following type objects to the layers shown:

<u>Object</u>	<u>Type(s)</u>	<u>Layer</u>
Solid	70	1
Lines and Arcs	3,5	41
Datum Planes and Axes	196, 197	61

(This Page Intentionally Left Blank)



Transformations

Lesson 11



Objectives

- Recognize situations where transformations can be used to assist in geometry creation.
- Recognize the process for performing transformations in a GRIP program.
- Demonstrate understanding of the types of transformations that can be performed in GRIP.

Introduction

The GRIP statements used to transform objects perform the same functions defined in Unigraphics using the Edit→Transform option.

Transformations are performed in GRIP using a matrix. There are three steps which must be followed as part of the transformation process, as described below:

1. Construct the geometry which will be the subject of the transformation.
2. Define the transformation matrix using the MATRIX command. This statement provides a variety of transformation operations which include: translation, scaling, mirroring, rotation, and combination (reposition).
3. Once the matrix is defined, the transformation is performed by using the matrix in the TRANSF major word.

GRIP does not allow transformation about a specific point or line in a single step. The scaling and rotation commands operate about the WCS origin or one of the principal axes, respectively. To perform these operations about another origin, it is necessary to use a series of GRIP statements. These operations will be described later in the chapter.

Matrices

A numerical matrix array is used by GRIP to process the transformation. This matrix array operates on a previously defined object. The matrix array must be declared as a numerical array with a dimension of 12 using the NUMBER/ statement. For example, the following declarations could be used to declare a matrix array:

```
NUMBER/ tr_mat(12), mat_list(4,12)
```

In the second declaration, mat_list is a two dimensional array which contains four matrices.

Graphic Image of a Matrix

1	0	0	0	X
0	1	0	0	Y
0	0	1	0	Z

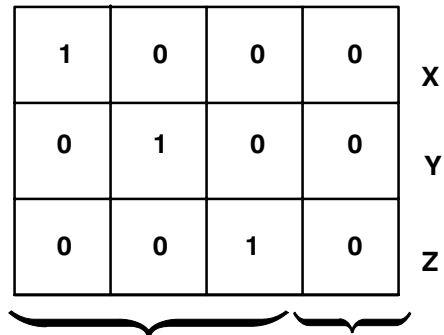


Figure 11–1 Identity or TRANSL/0,0,0 Matrix

Table 11–1 Transformation Commands

Function	Format
Transforms specified objects using the defined matrix.	TRANSF/matrix,object [,MOVE][,TRACRV]
Translates a delta distance	MATRIX/TRANSL,dx,dy,dz
Scales about the WCS origin	MATRIX/SCALE, {scale cx,cy,cz}
Mirrors about an existing line or plane.	MATRIX/MIRROR,line plane
Rotates about the ZC axis	MATRIX/XYROT,angle
Rotates about the XC axis	MATRIX/YZROT,angle
Rotates about the YC axis	MATRIX/ZXROT,angle
Combines 2 matrix operations The order is important.	MATRIX/matrix1,matrix2

TRANSF: Transformation



Synopsis

obj list = TRANSF/matrix,obj list[,MOVE][,TRACRV]

Description

Invokes various previously defined matrices that were created by the **MATRIX** statement. Execution of this statement can either cause new objects to be created, or the original objects to be moved, and provides a means for generating trace curves.

By default, the execution of this transformation creates a copy of the original objects and is object valued.

If you wish to move the original objects, you must use the minor word **MOVE**. In this case, do not use the **TRANSF** statement as an assignment.

Parameters

matrix

A previously defined matrix.

obj list

An existing object or an object list which may consist of object arrays.

MOVE

Minor word which indicates that the specified object is to be physically modified and that the modified object will remain on its original layer. If **MOVE** is not specified, the system makes a copy of the object, which then resides on the current work layer, and the original object remains unchanged.

TRACRV

Minor word which indicates that trace curves are to be generated as the object is moved. Trace curves, which are straight line segments, are generated by the end points of the curves. The trace curves will be straight lines even if the transformed objects are rotated.

Example

This example demonstrates the transformation of several objects using the **TRANSF** statement.

Trace curves are also created by specifying the **TRACRV** minor word.

Declarations

ENTITY/LN(8)
NUMBER/MAT1(12)

Geometry Definition

LN(1) =LINE/.25,.25,3,.25
LN(2) =LINE/3,.25,3,2
LN(3) =LINE/3,2,.25,2
LN(4) =LINE/.25,2,.25,.25

Matrix Definition

MAT1 = MATRIX/TRANSL,0,0,1

Transformation

LN(5..8)=TRANSF/MAT1,LN(1..4),TRACRV

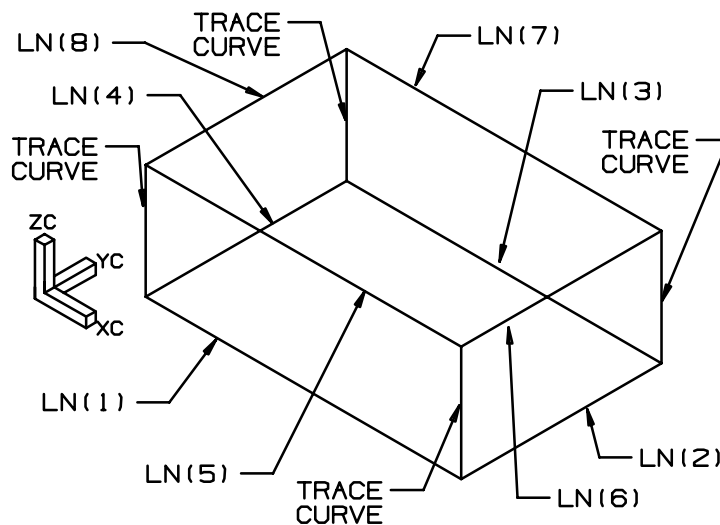


Figure 11–2 A transformation with trace curves



MATRIX: Translate



- Synopsis* **matrix = MATRIX/TRANSL,dx,dy,dz**
- Description* Defines a matrix used to perform a linear transformation of a delta distance in XC, YC, and ZC.
- Parameters* **TRANSL**
 Minor word which indicates that the values in the field which follows will define a matrix which represents a translation vector with work coordinates of dx, dy and dz.
- dx, dy, dz**
 The delta distances or displacement values in each of the three axes.
- Example* This example demonstrates the creation of a circle (CR2) by transforming a copy of CR1 using a translation matrix.
- Declarations* ENTITY/CR1,CR2
 NUMBER/MAT1(12)
- Geometry Definition* CR1 =CIRCLE/-1,-.5,.5
- Matrix Definition* MAT1=MATRIX/TRANSL,2,1,0
- Transformation* CR2 =TRANSF/MAT1,CR1

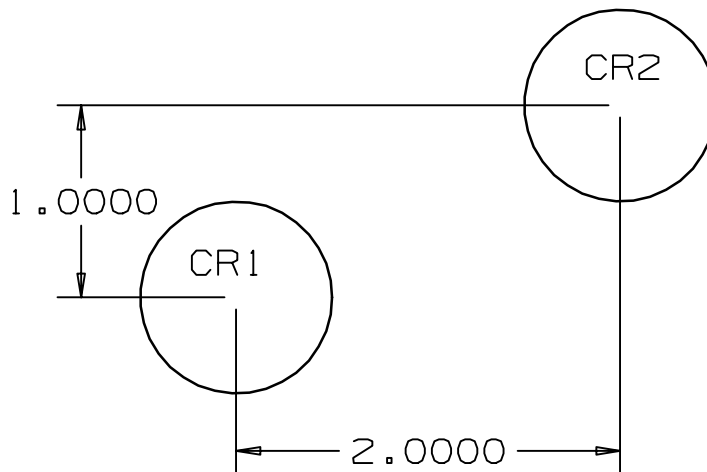


Figure 11–3 Translation matrix

MATRIX: Scale



<i>Synopsis</i>	matrix = MATRIX/SCALE{s ,xc,yc,zc}
<i>Description</i>	Defines a matrix which performs scaling about the origin of the work coordinate system. Non-uniform scaling may be applied to curves and B-surfaces only.
<i>Parameters</i>	<p>SCALE Minor word which indicates that the value in the field which follows represents a numerical scale factor by which a transformation can be performed.</p> <p>s The scaling factor which must be greater than zero. If the scale factor is less than one, the objects transformed will be reduced in size. If the scale factor is greater than one, the objects transformed will be enlarged.</p> <p>xc, yc, zc Allow you to apply different scale factors to the XC, YC, and ZC axes rather than using one scale factor for all three directions.</p>
<i>Example</i>	This example demonstrates the creation of two circles by transforming CR1 using a scaling matrix.
<i>Declarations</i>	ENTITY/CR1,CR2,CR3 NUMBER/MAT1(12)
<i>Geometry Definition</i>	CR1 = CIRCLE/0,0,.5
<i>Matrix Definition</i>	MAT1=MATRIX/SCALE,2
<i>Transformation</i>	CR2 = TRANSF/MAT1,CR1
<i>Matrix Definition</i>	MAT1=MATRIX/SCALE,.5
<i>Transformation</i>	CR3 = TRANSF/MAT1,CR1

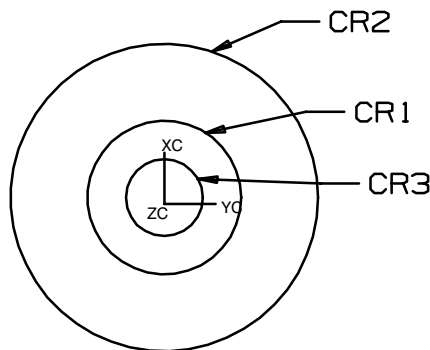


Figure 11-4 Scale matrix



MATRIX: Mirror

<i>Synopsis</i>	matrix = MATRIX/MIRROR, {<u>line</u> <u>plane</u>}
<i>Description</i>	Represents a reflection of objects through a plane. The plane may be normal to the current work plane by specifying a line containing the plane, or the plane may be a previously defined plane object anywhere in space.
<i>Parameters</i>	<p>MIRROR Minor word which indicates that the object in the field which follows will define a matrix through which a transformation may be performed.</p> <p><u>line</u> An existing line which lies in an imaginary plane. The imaginary plane is normal to the XY plane of the work coordinate system.</p> <p><u>plane</u> An existing plane.</p>
<i>Example</i>	This example demonstrates the creation of two lines by transforming two previously defined lines using a mirror matrix.
<i>Declarations</i>	ENTITY/LN0, LN1, LN2, LN3, LN4 NUMBER/MAT1(12)
<i>Geometry Definition</i>	LN0 = LINE/0, 1, 0, -1 LN1 = LINE/-1.5, .5, -.5, 0 LN2 = LINE/-1.5, -.5, -.5, 0
<i>Matrix Definition</i>	MAT1 = MATRIX/MIRROR, LN0
<i>Transformation</i>	LN3 = TRANSF/MAT1, LN1 LN4 = TRANSF/MAT1, LN2

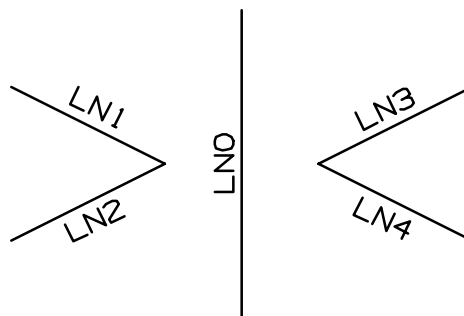


Figure 11–5 Mirror matrix

MATRIX: Rotate

Synopsis **matrix = MATRIX/{XYROT|YZROT|ZXROT},angle**

Description Represents the rotation in a principal plane of the work coordinate system about the axis normal to the plane. The principal planes are XY, YZ, and ZX.

Parameters **XYROT, YZROT, ZXROT**
 Minor words which indicate the principal plane in which rotation will take place.
 Rotation of objects takes place in the plane of the object, parallel to the specified principal plane, if the object or objects do not lie on that principal plane.

angle
 A numerical value which is the desired angle of rotation. A positive rotation is from the positive X axis to the positive Y axis for **XYROT**, from the positive Y axis to the positive Z axis for **YZROT**, and from the positive Z axis to positive X axis for **ZXROT**.

Example This example demonstrates the creation of a circle (CR2) by transforming CR1 using a rotation matrix in the XY principal plane.

Declarations ENTITY/CR1,CR2
 NUMBER/MAT1(12)

Geometry Definition CR1 =CIRCLE/1,-.5,.5

Matrix Definition MAT1=MATRIX/XYROT,45

Transformation CR2 =TRANSF/MAT1,CR1

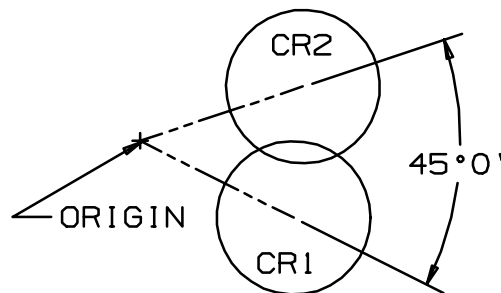


Figure 11–6 Rotation matrix

MATRIX: Multiple Transformations (Concatenation)



Synopsis **matrix = MATRIX/matrix1,matrix2**

Description Represents the product of two previously defined matrices. This matrix can be used to reposition objects, performing two transformations at the same time (e.g., translating and rotating, translating and scaling).

Parameters **matrix1,matrix2**
Two previously defined matrices which will be combined into one.

NOTE The resulting geometry will depend on the order in which the previously defined matrices are specified. Matrix 1 will be applied to the geometry first, followed by matrix 2.

Example This example demonstrates the creation of a series of mounting slots by transforming an existing slot using a reposition matrix. This example also demonstrates the use of a single numerical array and subscripted variable names for the matrices.

Declarations ENTITY/LN(2),CR(3),PT(2),GRP(6)
NUMBER/MAT(4,12)

Geometry Definition CR(1)=CIRCLE/0,0,4
LN(1)=LINE/2,.,25,3,.,25
LN(2)=LINE/2,-.25,3,-.25
PT(1)=POINT/2,0
PT(2)=POINT/3,0
CR(2)=CIRCLE/CENTER,PT(1),RADIUS,.,25,START,90,END,270
CR(3)=CIRCLE/CENTER,PT(2),RADIUS,.,25,START,270,END,90
GRP(1)=GROUP/LN,CR(2..3),PT(1),PT(2)

Matrix Definition MAT(1,1..12)=MATRIX/TRANSL,-1,0,0
MAT(2,1..12)=MATRIX/XYROT,45
MAT(3,1..12)=MATRIX/MAT(1,1..12),MAT(2,1..12)

Transformation GRP(2)=TRANSF/MAT(3,1..12),GRP(1)
GRP(3)=GROUP/GRP(1..2)

Matrix Definition MAT(4,1..12)=MATRIX/XYROT,90

Transformation

GRP(4)=TRANSF/MAT(4,1..12),GRP(3)
 GRP(5)=TRANSF/MAT(4,1..12),GRP(4)
 GRP(6)=TRANSF/MAT(4,1..12),GRP(5)

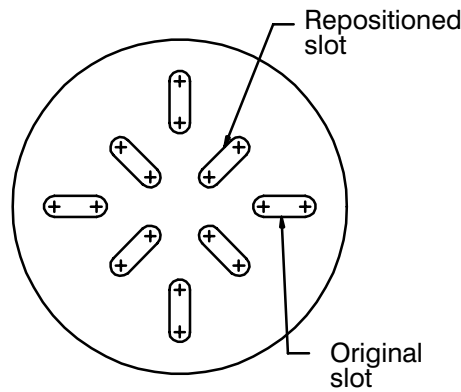


Figure 11–7 Repositioning of objects

Example

This example demonstrates the creation of two circles by transforming a previously defined circle using a reposition matrix. In this case, the object has been rotated and scaled.

Declarations

ENTITY/CR1,CR2,CR3
 NUMBER/MAT1(12),MAT2(12),MAT3(12)

Geometry Definition

CR1 =CIRCLE/0,.5,.25

Matrix Definition

MAT1=MATRIX/XYROT,-90
 MAT2=MATRIX/SCALE,2
 MAT3=MATRIX/MAT1,MAT2

Transformation

CR2 =TRANSF/MAT3,CR1

Matrix Definition

MAT1=MATRIX/XYROT,90
 MAT3=MATRIX/MAT2,MAT1

Transformation

CR3 =TRANSF/MAT3,CR1

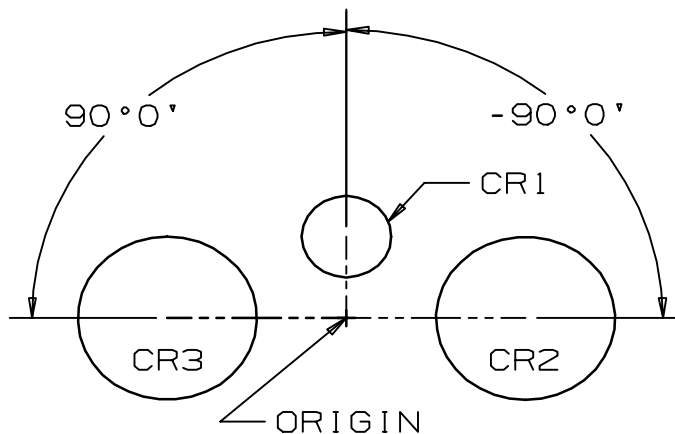


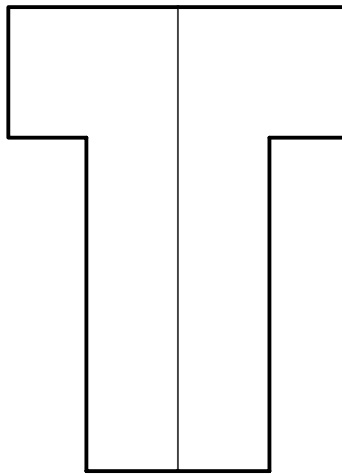
Figure 11–8 Multiplication matrices

Activity Transforming the L-Shape

This project will provide practice in using DO loops, transformations, and subscripted variables.

Follow these directions:

- Mirror the image created in the L-Shape activity about its leftmost vertical line to create a new shape that looks like a “T”. Use only one transformation command (TRANSF/). Use a DO loop and subscripted variables so that each new line that is created is assigned a unique variable name.



- Delete the vertical line that was used as the mirror line.
- Check for an active part in the beginning of you program.

Functions

Lesson 12

Objectives

- Demonstrate an understanding of the variety of mathematical, string, and entity functions available in GRIP.
- Recognize possible uses of these functions.



Processing Arithmetic Expressions

An *arithmetic expression* is the combination of constants, variables, and arithmetic operators. In an arithmetic expression, the operators are evaluated in the same order as in FORTRAN.

The hierarchy of these operators is shown below. The operators with the highest priority are at the top and the operators with the lowest priority are at the bottom. In cases where operators of equal priority appear in an expression, the system will evaluate the expression from left to right.

12

Priority	Operator	Meaning	Examples
Highest	**	Exponentiation	A**B
	/ *	Division & Multiplication	2/3, 2*3
Lowest	+ -	Addition & Subtraction	4+2
	=	Assignment	C=A-D

Parentheses

The order in which the operations in an expression are performed can be controlled using parentheses. Expressions within parentheses are evaluated before expressions without parentheses. If expressions in parentheses are nested within expressions in parentheses, the expression in the innermost set of parentheses is evaluated first.

Example Use of Parentheses

The use of parentheses in the evaluation of numerical expressions is shown below.

$$\begin{aligned}
 4+2*5-12/3 &= 10 \\
 (4+2)*5-12/3 &= 26 \\
 4+2*(5-12/3) &= 6 \\
 4+2*((5-12)/3) &= -2/3
 \end{aligned}$$

Mathematical Functions

The tables below describe the mathematical functions available in GRIP

Table 12–1 Arithmetic Functions

Format	Description
SQRTF(number)	Returns the square root of the number.
LOGF(number)	Returns the natural log of the number.
EXPF(number)	Returns “e” raised to the power of the number.
ABSF(number)	Returns the absolute value of the number.
INTF(number)	Returns the integer portion of the number.
MODF(num1,num2)	Returns the remainder of the division of num1 by num2.
MINF(num1,num2,...,numN)	Returns the smallest number.
MAXF(num1,num2,...,numN)	Returns the largest number.

Table 12–2 Trigonometric Functions

Format	Description
SINF(angle)	Returns the sine of the angle.
COSF(angle)	Returns the cosine of the angle.
ASINF(number)	Returns the angle whose sine is ‘number’.
ACOSF(number)	Returns the angle whose cosine is ‘number’.
ATANF(number)	Returns the angle whose tangent is ‘number’.



Standard mathematical functions are available in GRIP for arithmetic and trigonometric functions. These functions return either numbers or angles. Vector functions are not described in this manual. Mathematical functions are characterized as follows:

- The function names end with the letter F.
- The function operates on the number or expression within parentheses following the function.

Example

The examples below are all valid uses of SQRTF function:

```
D = SQRTF(9)
dia = 4*(SQRTF(18+63) + 2*r)
c = SQRTF(a**2 + b**2)
```


Example INTF

A=INTF(4.9979) A equals 4

MODF returns the remainder of the division arg/mod. The value returned will be in the base of the denominator “mod”. Therefore, for most practical applications the value mod should be used as an integer.

B=MODF(X, Y)

is equivalent to the following expressions:

A=X/Y
 B=ABSF(A-INTF(A))*Y

Example MODF

B=MODF(14, 8) B equals 6

num1=7.5
 num2=3
 C=MODF(num1, num2) MODF equals 1.5

MINF returns the smallest numerical value in the argument list to a receiving variable or GRIP statement. This statement is limited to 102 arguments.

Example MINF

SMALL=MINF(-1, 2, 4, 10, 7) SMALL equals -1

MAXF returns the largest numerical value in the argument list to a receiving variable or GRIP statement. This statement is limited to 102 arguments.

Example MAXF

BIG=MAXF(-1, 2, 4, 10, 7) BIG equals 10



Trigonometric Functions

The GRIP trigonometric functions (SINF, COSF, ACOSF, and ATANF) are listed in Table 12–2.

SINF, the sine function, returns the sine of the argument to a receiving variable or GRIP statement. The returned value will be between -1 and 1 inclusive. The range of the specified angle in degrees is infinite.

Example SINF

SIN = SINF(95)	SIN equals .9962
SIN = SINF(405)	SIN equals .7071

In the examples above, SIN will equal the sine of the angle (i.e., 95° , or 405°). The angle is in degrees, but may be a number, numeric variable, or expression.

COSF, the cosine function, returns the cosine of the argument to a receiving variable or GRIP statement. Although the returned value will be between -1 and 1 inclusive, the range of the specified angle in degrees is infinite.

Example COSF

COS = COSF(95)	COS equals $-.0872$
COS = COSF(405)	COS equals .7071

In the examples above, COS will equal the cosine of the angle (i.e., 95° , or 405°). The angle is in degrees, but may be a number, numeric variable, or expression. Remember, tangent equals sine divided by cosine:

$$C = \text{SINF}(\text{angle}) / \text{COSF}(\text{angle})$$


ASINF, the arcsine function, returns an angle for which the argument is the sine to a receiving variable or GRIP statement. The returned value will be between -90 and 90 degrees inclusive.

Example ASINF

ANG = ASINF(-.5) ANG equals -30

ACOSF returns an angle for which the argument is the cosine to a receiving variable or GRIP statement. The returned value will be between 0 and 180 degrees inclusive.

Example ACOSF

ANG = ACOSF(.5) ANG equals 60

ATANF returns an angle for which the argument is the tangent to a receiving variable or GRIP statement. The returned value will be between -90 and 90 degrees inclusive.

Example ATANF

ANG = ATANF(2.5) ANG equals 68.1986



String Functions

GRIP provides you with two types of string functions: string-valued and number-valued. String-valued functions return character strings and number-valued functions convert strings into numbers. Each are briefly described below.

String-valued Functions

Character valued functions either generate or manipulate characters in a string. For example, the function BLSTR creates blank spaces in a string. Since blanks are a form of a text character, BLSTR is character valued.

DATE
TIME
BLSTR(n)
CHRSTR(n)
DFSTR(n)
FSTR(n)
ISTR(n)
REPSTR(s1,s2,s3,p)
SUBSTR(s,p,c)

Number-valued Functions

Number valued functions return numerical values concerning characteristics pertaining to the specified characters. For example, the function ASCII returns the ASCII equivalent of a specified character string in decimal form. Since GRIP considers this a number, ASCII is a number valued function.

ASCII(s,p)
FNDSTR(s1,s2,p)
LENF(s)
VALF(s)



Return Current Date

Synopsis

str=DATE

Description

Returns the current date as a ten or twelve character literal string. The format of the string is set by &FULLDT (see the following page). &FULLDT supports the following formats:

mm--dd--yy
 mm--dd--yyyy
 mm-dd-yyyy

If you do not set &FULLDT, the DATE command defaults to the following format:

mm--dd--yy

Example

This example demonstrates the three formats available using the &FULLDT GPA.

```
DELIM/ '
&FULLDT = 0
print/ &FULLDT = ',&FULLDT
print/ DATE = '+DATE
print/ '
```

```
&FULLDT = 1
print/ &FULLDT = ',&FULLDT
print/ DATE = '+DATE
print/ '
```

```
&FULLDT = 2
print/ &FULLDT = ',&FULLDT
print/ DATE = '+DATE
HALT
```

The program produces the following results:

```
&FULLDT = .0000
DATE = 08--21--98
```

```
&FULLDT = 1.0000
DATE = 08--21--1998
```

```
&FULLDT = 2.0000
DATE = 08-21-1998
```



Set Full Date Flag

Synopsis

&FULLDT

Description

Enables the DATE command to output the full date (mm--dd--yyyy). The full date includes the century portion of the date when &FULLDT is set to 1 or 2. Otherwise, the DATE command outputs the year without the century (mm--dd--yy).

Characteristics

Read/Write Number [0..2]

Constant Values

You can set &FULLDT to 0, 1, or 2. These values correspond to the following formats:

- 0 = mm--dd--yy
- 1 = mm--dd--yyyy
- 2 = mm--dd--yyyy

Example

```
&FULLDT=1
PRINT/DATE
&FULLDT=0
PRINT/DATE
HALT
```

If the date were July 6, 1999, then the above program would produce:

```
07--06--1999
07--06--99
```



Return Current Time

Synopsis

str=TIME

Description

Returns the current time as a five character literal string HH:MM.

Example

The following example demonstrates the use of the **TIME** function.

STRING/STR(32)

Function	Assigned Value
STR='The time is '+TIME	STR=The time is 12:45

12

Create Blank Characters

Synopsis

str=BLSTR(n)

Description

Generates the number of blank characters, spaces, specified by the value n. The value of n must be from 0 to 132 inclusive.

Parameters

n

The number of characters to be created. The number must be a positive integer value between 0 and 132 inclusive.

Example

This example demonstrates the creation of a string with blank characters.

```
STRING/STR(4,15)
STR(1)='X'+BLSTR(15)+'Y'
PRINT/STR
```

The following string would be printed on the CRT:

```
X                Y
```

BLSTR can be imbedded in a string (as above) or assigned directly to a variable which will space a table of values as follows:

```
S=BLSTR(10)
STR(2)='2.5'+S+'1.25'
STR(3)='3.6'+S+'1.25'
STR(4)='6.2'+S+'2.27'
PRINT/STR
```

The statements above would create the following table on the CRT:

```
2.5                1.25
3.6                1.25
6.2                2.27
```



Number of Characters in a String

Synopsis **num=LENF('string')**

Description Returns the number of characters in a string. A string may be declared as containing a maximum of 132 characters, but the actual number of characters in the string may be less than 132. Leading and trailing as well as imbedded blanks are counted.

Parameters **'string'**
 The literal string or string variable name of a string whose length in characters will be returned.

Example This example demonstrates obtaining the length of several strings.

STRING/ABC(10)

Function	Assigned Value
ABC='MESSAGE'	
ANS=LENF(ABC)	ANS=7
ANS=LENF(' XY ')	ANS=2



Convert Integer to Character String

Synopsis

str=ISTR(n)

Description

Converts the integer portion of a real number into a character string. The maximum number of characters which this function can return is eight and the value of n must be any valid Unigraphics number. To return more than eight characters, use the **ISTRL** statement.

Parameters

n

Real number whose integer portion will be converted to a character string. The non–integer (to the right of the decimal point) portion will be truncated.

Example

This example demonstrates the use of the **ISTR** function to convert the integer portion of a number to a string.

```
ENTITY/ent(1000)
NUMBER/n

MASK/70
IDENT/'Select Solids to Delete',sol,CNT,n,resp

PRINT/'Solids Selected = '+ISTR(n)
```

The following text will be printed to the listing window:

Solids Selected = 13



Convert Real Number to Character String

Synopsis

str=FSTR(n)

Description

Converts a real number into a character string. The maximum number of characters which this function can return is eight, including the decimal point and minus sign. To return more than eight characters, use the **FSTRL** statement.

The resulting string has a leading zero when the number is less than one, but greater than negative one. You may use **&DECPL** to control the number of decimal places that appear to the right of the decimal point.

NOTE Please note the following items.

- This function rounds up rather than truncating digits when specifying less than the significant number of digits with **&DECPL** or with the default number of decimal places if **&DECPL** is not used. For example, if you set **&DECPL=0**, then **FSTR(2.5)** rounds up to 3.
- If you set **&DECPL = 0**, the decimal point does not return in the string.

Parameters

n

Real number which is to be converted to a character string. This function returns a maximum of 8 characters (including the decimal point and minus sign). If you attempt to convert a real number which generates more than eight characters, then this command returns a string of nines (e.g. '9999999') which indicates an overflow error.

Example

This example demonstrates the creation of a note which contains the radius of a circle.

```
ENTITY/CR1
NUM=2.5
CR1=CIRCLE/0,0,NUM
&DECPL=4
NOTE/0,-3,'THE RADIUS = '+FSTR(NUM)
HALT
```

The NOTE reads as follows:

```
THE RADIUS = 2.5000
```



Convert String to Real Number

Synopsis

num=VALF('string')

Description

Converts a string representing a numerical value into a real floating point number. The object string must be in the numeric form with a maximum of 132 characters including the decimal point. This function is the reciprocal of the **FSTR** and **FSTRL** functions.

Parameters

'string'

The literal string or string variable name of a string which will be converted to a real number.

Example

This example will create a table of text on the screen which represents a set of standard radii used on a drawing. The user can then select the text to define the circle radius.

```
ENTITY/CR1
STRING/STR(8)
DATA/X,0,Y,0,Z,0
```

```
STR='0.375'
NUM=VALF(STR)
```

```
CR1=CIRCLE/X,Y,Z,NUM
```



ASCII Value of a Character

Synopsis **num=ASCII('string',pos)**

Description Returns the ASCII value of the character specified by pos, in decimal form. Refer to the ASCII Conversion Table for a complete list of ASCII characters and their corresponding decimal values. If pos is less than one or greater than the number of characters in the string, the error message INVALID CHAR POS will be displayed.

Parameters **'string'**
 The literal string or string variable name of the string to be accessed.

pos
 The character position of the character whose ASCII value will be returned. Spaces (blanks) are counted in arriving at the position. The pos must be an integer value greater than zero.

Example This example demonstrates the use of the ASCII function to capitalize a name.

```

STRING/ usr(20)
NUMBER/ v

BCK010:
    TEXT/'Enter Last Name',usr,resp
    JUMP/BCK010:,CANCEL:,resp

    v = ASCII(usr,1)        $$ Get ASCII value of first character

    IFTHEN/v>=97 AND v<=122
        len= LENF(usr)
        usr = CHRSTR(v-32)+SUBSTR(usr,2,len-1)
    ENDIF

CANCEL:
    HALT
    
```



Return string with ASCII Value of N

Synopsis

str=CHRSTR(n)

Description

Returns a single character string that has an ASCII value equal to the value n to a receiving variable or GRIP statement. The receiving variable must be declared as a string.

Parameters

n

An integer which represents a the ASCII value of the character to be returned. The value of n must be from 32 to 126 inclusive, as indicated in the ASCII conversion table.

Example

This example demonstrates using the **CHRSTR** function to capitalize a letter.

STRING/ rev(1)
NUMBER/v

rev = 'b'

v = ASCII(rev,1) \$\$ Returns ASCII value of character
IF/v>=97 AND v<=122, rev = CHRSTR(v-32)

The resulting value of rev will be “B”. The number val was assigned the ASCII value of “b” (98). Thirty two was subtracted from n (66) and converted to a string (“B”) and assigned to rev.



Replace Characters in a String

Synopsis

**str=REPSTR('object string','search string',
'replacement string',pos)**

Description

Replaces a specific character string with another. In this function the object string is the string to be altered, the search string is the string to be found and the replacement string the new string. The pos value specifies a position in the string where the search is to begin. The search will continue to the last position in the string or until the first occurrence of the search string is found. If the search string is not found, the object string is returned unchanged. If pos is less than one or greater than the number of characters in the string, the error message INVALID CHAR POS will be displayed.

Parameters

'object string'

The literal string or string variable name of the string to be altered.

'search string'

The set of character(s) which will be searched for in the specified object string. Only the first occurrence of the search string, past the starting position (pos), will be replaced.

'replacement string'

The set of character(s) which will replace the search string characters. The number of characters in the search and replacement strings do not have to be equal.

pos

Character position in the object string where the search will begin. Pos must be an integer value greater than zero. Spaces (blank characters) are counted in determining a character's position in the object string. The positions are counted from left to right with the first character in the string being 1.



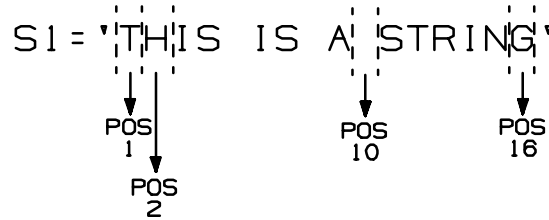


Figure 12–1 Numeric positions of characters in a string

Example

This example demonstrates the process of replacing several strings.

STRING/STR1(10),STR2(10)

Function	Assigned Value
STR1='6--09--99'	STR1= 6--09--99
STR2=REPSTR(STR1,'--','/',1)	STR2= 6/09--99
STR2=REPSTR(STR2,'--','/',1)	STR2= 6/09/99
STR2=REPSTR(STR2,'0','"',3)	STR2= 6/9/99



Extract a portion of a Character String

Synopsis **str=SUBSTR('object string',pos,count)**

Description Extracts a specified portion of a character string. The starting position in the object string is specified by the value pos and the number of characters to be extracted is specified by the value count. If the character count exceeds the number of characters in the object string, the substring which is returned will be the object string.

Parameters **'object string'**
 The literal string or string variable name of the string from which the specified characters will be copied.

pos
 Character position in the object string where the copying of characters will begin. Spaces (blanks) are counted in arriving at the starting position. The pos must be an integer value greater than zero.

count
 The number of consecutive characters to be copied. Spaces (blanks) are counted in the count. The count must be an integer value greater than zero.

Example This example demonstrates the use of the **SUBSTR** function to copy characters from a text string.

STRING/STR1(32),STR2(32)

Function	Assigned Value
STR1='THIS IS A STRING'	STR1=THIS IS A STRING
STR2=SUBSTR(STR1,11,6)	STR2=STRING



Position of Characters in a String

Synopsis

num=FNDSTR('object string','search string',pos)

Description

Returns the start position of a search string within an object string. The returned value is an integer which represents the start position of the first occurrence of the search string. The search will be made from the position specified by pos to the end of the object string or until the first occurrence of the search string is found. If the search string is not found, a 0 is returned.

Parameters

'object string'

The literal string or string variable name of a string which will be searched.

'search string'

The set of character(s) which will be searched for in the object string.

pos

Position in the object string where the search will begin. Spaces (blanks) will be counted in arriving at the search location. The pos value must be an integer greater than zero.

Example

This example demonstrates the use of the **FNDSTR** function to find where the characters "83" are in a string. Only an occurrence after the seventh character will be searched for.

```
STRING/STR(15)  
STR='54738378332'  
ANS=FNDSTR(STR,'83',7)
```

The variable ANS is assigned a value of 8 in the example above. Although the first occurrence of the search string, 83, is at position 5, it was ignored because 7 was specified as the start search position.

Activity: Date Conversion

- Write the code to convert the date from the format returned by the **DATE** function to the following format:

DD MMM YYYY

where MMM is a three letter mnemonic for the month as listed below:

JAN	FEB	MAR
APR	MAY	JUN
JUL	AUG	SEP
OCT	NOV	DEC



Activity: Case Conversion (Optional)

- Write a series of statements to convert a string of any length from lower case to upper case.

By referring to the ASCII Conversion Chart in Appendix B, notice the arrangement of the lower case characters and the upper case characters in the table, along with their corresponding ASCII values.

You will need a loop to analyze each character, to determine whether a conversion is necessary.



12

(This Page Intentionally Left Blank)

Subroutines

Lesson 13

Objectives

- Decide when to use a subroutine in a GRIP program.
- Pass parameters to a subroutine.
- Use the statements for subroutines; CALL, PROC, and RETURN.
- Navigate the subroutine file organization with the GRIP search list.



Subroutine Programming

A subroutine is a series of GRIP statements that perform a specific task under the control of a main program. The subroutine has the following features:

- Subprograms can be executed as often as needed by calls from the main program or other subprograms.
- Subprograms are separate source files that are compiled individually.
- Subprograms may call other subprograms (50 max).
- Subprograms cannot be recursive.

Subroutines have the same structure as a main program, but must contain two statements that distinguish it from the main program. A main program cannot contain either of these statements.

- **PROC** statement. This stands for 'procedure' and must be the first statement in a subroutine. The **PROC** statement includes a list of arguments shared between the subroutine and the calling program.
- The **RETURN** statement signals the end of a subroutine.

To use a subroutine, a third statement is needed in the calling program to invoke the subroutine.

- The **CALL** statement invokes the subprogram. This statement can be used in the main program or in another subprogram.

Table 13–1 Subroutine Programming Statement Formats

Function	Format
Call subprogram	CALL/'subprogram name'[,argument list]
Subroutine start	PROC[/argument list]
End of subroutine	RETURN

13

Call Statement

Synopsis

CALL/'subprogram name',[argument list]

Description

The **CALL** statement makes a subprogram available for use. When the **CALL** statement is encountered in a program, control of execution is transferred to the first executable statement of the subprogram. When execution in the subprogram encounters a **RETURN** statement, control is returned to the calling program at the statement immediately following the **CALL** statement. The 'subprogram name' is the name under which the subprogram is stored in the GRIP object library (no file extension is needed). The name used must be a string literal; string variables are not allowed. The "argument list" is optional and can contain up to 25 variable names which may be any combination of numbers, entities, strings or variables. A variable may be either a simple variable or an array of which all or only a portion may be passed to the subprogram. Passing only a portion of an array, however, involves the use of the subrange operator and may only be used on the last (right most) variable in the list.

NOTE

GRIP allows a total of 1000 labels per subroutine. A **CALL** command generates one label. This label counts against the total you can use.

Example

The following statement, for example, is valid because the array **B** is the last argument in the list.

```
CALL/ 'SUB1' , X , Y , A , B ( 10 . . 25 )
```

The following statement, however, would have resulted in an error.

```
CALL/ 'SUB1' , X , Y , A , B ( 10 . . 25 ) , C
```

An alternative to using the subrange operator in a **CALL** statement is to pass the entire array and the subrange bounds as two additional arguments. However, whether a single value, an array, or a portion of an array is passed in the argument list, the corresponding dummy arguments in the subprogram must agree in type, quantity, and order.

Procedure Statement

Synopsis

PROC[/argument list]

The PROC statement must be the first statement in any subprogram. The optional argument list can consist of up to 25 variable names. These are local variable names and are known only to the subprogram where they appear. A subprogram may contain any number of source statements and may contain any GRIP statement.

If values are to be passed from the calling program to the subprogram, there must be a dummy argument in the argument list for each value or array. In addition, the dummy arguments must be of the same type and quantity as those being passed by the calling program.

Except for simple numerical values and constants, all dummy arguments must be declared. Entity and number declarations are the same as in a main program, however, only the number parameters may be declared for string variables which are in the argument list.

The variables in a subprogram, including those in the dummy argument list, are considered local variables and are known only to the subprogram. Therefore the same variable names may be used in the main program and individual subprograms without conflict. It should be noted, however, that it is the location in the argument and not the variable name, that determines what the assignment will be. For example, if the call statement “CALL/SUB1,X,Y,Z” was issued and the PROC statement in the subprogram was “PROC/Z,Y,X”, the variable assignment in the subprogram would be $Z=X$, $Y=Y$, and $X=Z$.

Example

In this example, a calling program will pass a simple numerical variable, three elements of a numerical array, an entity, an entity array with four elements, a string variable with a character length of ten and a string array with ten elements and a character length of twenty.

The calling program may appear as:

```

ENTITY/LN1 , PT( 4 )
NUMBER/NUM( 5 ) , VAL1
STRING/ABC( 10 ) , XYZ( 10 , 20 )
.
.
.
CALL/ ' SUB1 ' , VAL1 , LN1 , PT , ABC , XYZ , NUM( 1 . . 3 )
LINE/- - - - $$ ADDITIONAL LINES OF CODE
.
.
.
HALT

```

The subroutine SUB1 may appear as:

```

PROC/ INDX , LN1 , PERIM , TITL , XYZ , NUM
ENTITY/LN1 , PERIM( 4 )
NUMBER/NUM( 3 ) , INDX
STRING/TITL , XYZ( 10 )
.
.
.
RETURN

```

Notes:

- VAL1 (equivalent to INDX in SUB1) is a simple numerical variable. In the subroutine, NUM was declared as an array of three instead of five because a subrange operator was used to pass only three numbers. Note that it must be placed last in the argument list.
- The entity variables LN1 and PT (equivalent to PERIM in SUB1) were declared the same as they were in the calling program.
- The string variable ABC (equivalent to TITL in SUB1) was declared as containing a maximum of ten characters in the main program but TITL must not have a size associated with it in the subroutine declarative because it is a singular string. XYZ, however, was declared with a length of ten because that is the number of string elements in the XYZ array.

Return Statement

Synopsis

RETURN

The RETURN statement may only appear in a subprogram. A subprogram may contain more than one RETURN statement; however, when one is encountered control is immediately returned to the calling program. The dummy arguments in the subprogram will be passed to the actual arguments in the calling program and the statement following the CALL statement in the calling program will be executed.

Organizing Subroutines using the “gri.sea” File

Each subroutine resides in a separate source file (.grs extension) with a corresponding object file (.gri extension). When many GRIP programs are written, there may be many GRIP source and object files. If all of these files are placed in the same directory, it may become confusing to determine which files belong to each program. It is therefore desirable to organize files in different directories.

Object files can be linked together even if they reside in different directories. This is accomplished by using search list. The search list is a text file named “gri.sea” which consists of directory specifications (full path names) where the object files reside.

When the linker operates, it first searches for object code in the current directory as specified in the GRADE session. If an object file is not found, the “gri.sea” file (if one exists in the current directory) is opened. The linker sequentially searches for .gri files in each directory specified in “gri.sea”.



Activity: Case Converter Subroutine

- Write a subroutine to convert all characters in a string to upper case or to lower case characters. Test the program by writing a main program to prompt for a string, then convert the string to upper case and to lower case characters. The new strings should then be displayed on the listing window.

NOTE When developing your **case_converter** subroutine, prefix your subroutine with your initials, to distinguish your code with that of other students.

Typical call statements to the subroutine would look like the following:

```
CALL/ 'xxx_case_converter', $
      str,                $ String to convert
      1,                  $ Convert to upper
      rc                  $$ Return code

CALL/ 'xxx_case_converter', $
      str                 $ String to convert
      2,                  $ Convert to lower
      rc                  $$ Return code
```

An overall outline of the **xxx_case_converter** subroutine might look like:

```
PROC/                                $
      str,                            $ String to convert
      cnvrt_typ,                       $ Convert type
      rc                                $$ Return code
STRING/ str
NUMBER/ cnvrt_typ, rc, $
        UPPER, LOWER, SUCCESS, FATAL
DATA/ UPPER,1, LOWER,2, SUCCESS,0, FATAL,-1

rc = SUCCESS

IFTHEN/ cnvrt_typ == UPPER

      (convert str to upper case)
```

```
ELSEIF/ cnvrt_typ == LOWER
    (convert str to lower case)

ELSE/
    rc = FATAL          $$ Invalid type
ENDIF

RETURN
```

(This Page Intentionally Left Blank)

13

Drafting Functions

Lesson 14

Objectives

- Gain a broad understanding of the commands available to create and edit drawings.
- Learn the commands to import part files and drawing borders.

The following commands control the creation, editing, and deleting of drawings. Also included are the commands to import drawing borders on to a drawing.

Table 14–1 Drawing Commands

Function	Format
Create a Drawing	DRAWC/'drawing name',[MMETER.] {height,width n}[,IFERR,label:]
Change Drawing Size	DRAWE/'drawing name',]SIZE, [{INCHES MMETER},]{height, width n}[,IFERR,label:]
Add View to a Drawing	DRAWE/'drawing name',]ADD,'view name',x,y[,IFERR,label:]
Verify a Drawing	DRAWV/'drawingname',] [,{PLOT DVSTAT,variable,}]variable list[,IFERR,label:]
Delete a Drawing	DRAWD/'drawing name '[,IFERR,label:]
Current Drawing	&CURDRW
Import a Part	RPATT/'filespec'[{,matrix csys,scale}] [,LAYER][,PLMODS,value] [,NOVIEW][,RETCAM,number] [,IFERR,label:]
Import a Part Grouped	obj = RPATTG/'filespec' [{,matrix csys,scale}][,LAYER] [,PLMODS,value][,NOVIEW] [,RETCAM,number][,IFERR,label:]



Drawing Overview

This section covers the GRIP statements used to produce drawings. Drawings can be created, edited, verified, deleted, and renamed. As drawings are created, they are automatically saved. Use the **GPA &CURDRW** to retrieve a specified drawing.

Specifying Drawing Member View Names

When a drawing member view name or drawing work view name is specified, it needs to be appended with an “@” character to denote it as a drawing member view or drawing work view. Specification of the view name this way is actually a short cut since the actual view name contains an integer following the “@” character (for drawings that integer is zero). Optionally, you can enter the full name in order to assure uniqueness of member view names (drawings can have more than one instance of a given view on a single drawing).

An example of a drawing member view name that was in some way derived from the model view, TOP, is “TOP@3”. To specify the name of the model view, “TOP” should be used. To specify the name of the drawing member view, either “TOP@” or “TOP@3” can be used. The full names of all the member views for a given drawing can be retrieved using **DRAWV**.

NOTE In Unigraphics, when the drawing view is the work view, the layout which contains the drawing is called !DRAWING. This layout name is read only and cannot be used to set the current layout.

Must Be in Drafting to Use Drawings

When running interactive GRIP, functions that create, retrieve, or require the presence of a drawing are only allowed in the following modules: Gateway, Drafting, Valisys, and Genconnect. Those functions are also available in GRIP Batch.

When drafting objects are created on a drawing in GRIP, the objects associated to the drafting object (the dimensioned objects) must be objects that are on the drawing.

Create a Drawing

Synopsis

**DRAWC/'drawing name',[MMETER,]{height,width | n}
[,IFERR,label:]**

Description

Creates a drawing of a specified name and size in the database. The specified drawing becomes the current drawing. Once the drawing is created, views can be added and manipulated using the Edit Drawing statement, **DRAWE**.

Parameters

'drawing name'

A string or string variable which represents the name of the drawing. You must use this name to refer to the drawing for any other purpose (e.g., editing, placing in a layout, etc.).

MMETER

Minor word which indicates that the drawing size is to be specified in millimeters. Inches is used if this minor word is not specified.

height,width

The size of the drawing to be created. Height is measured in the positive YC axis and width is measured in the positive XC axis. The drawing is created with its lower left hand corner at the origin of the WCS.

n

A valid drawing size code. The code creates the drawing in either inches or millimeters as specified by use of the minor word **MMETER**. The valid codes are as follows:

Code	Size in Inches	Size in Millimeters
1	Ansi A – 8.5 x 11	ISO A0 – 841 x 1189
2	Ansi B – 11 x 17	ISO A1 – 594 x 841
3	Ansi C – 17 x 22	ISO A2 – 420 x 594
4	Ansi D – 22 x 34	ISO A3 – 297 x 420
5	Ansi E – 34 x 44	ISO A4 – 210 x 297

IFERR,label:

Specifies a label to which the program jumps if an error occurs in creating the drawing.



Change Drawing Size

Synopsis

DRAW/'drawing name',]SIZE, [{INCHES | MMETER},]
{height,width | n} [,IFERR,label:]

Description

Changes the size of the drawing format of a drawing. Drawing units are a parameter of the drawing. Therefore, if no unit minor word is specified in this statement but a drawing code or size is, then the current parameter applies.

Parameters

'drawing name'

A string or string variable which represents the name of the drawing whose drawing format size is to be changed.

SIZE

Minor word which indicates that the drawing size is to be changed.

INCHES

Minor word which indicates that the drawing size is specified in inches.

MMETER

Minor word which indicates that the drawing size is specified in millimeters.

height,width

The new size of the drawing format. Height is measured in the positive YC axis and width is measured in the positive XC axis. The drawing is created with its lower left hand corner at the origin of the ABS.

n

A valid drawing size code. The code creates the drawing format in either inches or millimeters as specified by use of the minor words **INCHES** and **MMETER**.

Code	Size in Inches	Size in Millimeters
1	Ansi A – 8.5 x 11	ISO A0 – 841 x 1189
2	Ansi B – 11 x 17	ISO A1 – 594 x 841
3	Ansi C – 17 x 22	ISO A2 – 420 x 594
4	Ansi D – 22 x 34	ISO A3 – 297 x 420
5	Ansi E – 34 x 44	ISO A4 – 210 x 297

IFERR,label:

Specifies a label to which the program jumps if an error occurs in editing the drawing.

Add View to Drawing

Synopsis

**DRAWE/['drawing name',]ADD,'view name',x,y
[,IFERR,label:]**

Description

Adds an existing view to a previously created drawing at the specified coordinates.

Parameters

'drawing name'

A string or string variable which represents the name of the drawing to which the view is added.

ADD

Minor word which indicates that a view is to be added to a drawing.

'view name'

A string or string variable which represents the name of the view to be added. The view must not already be on the drawing and must not be a drawing.

x,y

The coordinates which define the drawing reference point and are used to place the view in the drawing. The coordinates are measured from the lower left hand corner of the drawing to the *view reference point* of the view being added.

IFERR,label:

Specifies a label to which the program jumps if an error occurs in editing the drawing.



Verify a Drawing

Synopsis

**DRAWV/['drawing name',][{PLOT|DVSTAT,variable,}]
variable list[,IFERR,label:]**

Description

Returns the parameters of the specified drawing to a series of variables. Some of the variables in the list must be declared as arrays. The parameter returned depends upon its field position in the list. Therefore, if certain parameters are not desired, the field must be defined using two commas (,,). However, once you have specified the last desired parameter, do not use commas to define the remaining positions. For example, if you wished to only verify the name of each member view and the number of member views, you would specify

DRAWV/VNAM,,NUMVW,IFERR,label:

where VNAM is the member view name; ,, represents the omitted reference coordinates parameter, and NUMVW represents the number of member views.

You can verify either plot or non-plot information. This is determined by the minor word **PLOT**. If verifying non-plot information (by omitting the minor word **PLOT**), you can also return the status of each view in addition to the regular list of data. This is done by specifying the minor word **DVSTAT** followed by a numerical array.

NOTE You cannot specify both **PLOT** and **DVSTAT** in the same statement.

If you do not specify the minor word **PLOT**, the following regular list of data is returned to a variable list:

Position	Parameter	Data Type
1	Name of each member view	String array
2	Reference point (x,y) of each member view in drawing coordinates	Num. array
3	Number of member views	Number
4	Drawing units Inches Millimeters	[1,2] 1 2
5	Drawing height	Number
6	Drawing width	Number

If you specify the minor word **PLOT**, the following plot information is returned to a variable list:

Position	Parameter	Data Type
1	Plotter name	String Variable
2	Scale	Number
3	Rotation angle	Number
4	Media reference number	Number
5	Number of copies	Number
6	X coordinate of plot origin (in drawing coordinates)	Number
7	Y coordinate of plot origin (in drawing coordinates)	Number
8	X offset value from the plotter origin	Number
9	Y offset value from the plotter origin	Number
10	Pen assignment By density By color	[1,2] 1 2
11-25	Pen list	Number

Parameters

'drawing name'

A string or string variable which represents the name of the drawing being accessed. If no name is specified, the current drawing is accessed. If no drawing name is specified and there is no current drawing, an error occurs.

PLOT

Minor word which indicates that the plotting parameters of the drawing are to be returned to the variable list.

DVSTAT

Minor word which indicates that the status (reference or active) of each view on the drawing is to be returned.

variable

A numerical array which receives the status of each view in the drawing when the minor word **DVSTAT** is specified. The view status is returned in the same order as the view names which can be extracted in this and other **DRAWV** statements.



The values returned are:

0 for a Reference View

1 for an Active View

variable list

A list of variables to which either plot or non-plot parameters are assigned. Some of the variables in the list must be declared as arrays. Specifying the minor word **PLOT** returns plot information to this list. Omitting **PLOT** returns non-plot data.

IFERR,label:

Specifies a label to which program execution jumps if an error occurs while the drawing is being accessed.

Example

This example demonstrates the use of the Verify Drawing statement **DRAWV** to return the parameters of a drawing.

Declarations

```
NUMBER/REF(10,2),SIZE(2)
STRING/VW(10,30)
DRAWC/'DRAW1',4,WORK,IFERR,L10:
DRAWE/ADD,'FRONT',2,3
DRAWE/ADD,'TOP',2,8
DRAWE/ADD,'TFR-ISO',8,6
DRAWV/VW,REF,,NUM1,SIZE,IFERR,L20:
L10:MESSG/'CANNOT CREATE DRAWING'
L20:MESSG/'CANNOT VERIFY DRAWING'
```

The number variables do not need to be declared; however, the string array which holds the view names and the numerical array which holds the reference point locations must be declared. The variables are assigned values as follows:



Variable	Value Assigned
VW(1)	FRONT
VW(2)	TOP
VW(3)	TFR-ISO
REF(1,1)	2
REF(1,2)	3
REF(2,1)	2
REF(2,2)	8
REF(3,1)	8
REF(3,2)	6
NUM1	1
SIZE(1)	22
SIZE(2)	34

The number of member views was not required; therefore, the field (third field) was left blank by using back to back commas (,,).

NOTE The **DRAWE** and **DRAWV** statements do not use the drawing name because the drawing is active in the current work view.

Delete a Drawing

<i>Synopsis</i>	DRAWD/'drawing name',[IFERR,label:]
<i>Description</i>	Deletes the specified drawing.
<i>Parameters</i>	'drawing name' A string or string variable representing the name of a drawing to delete. IFERR,label: Specifies a label to which the program jumps if an error occurs. Possible errors include: invalid drawing name; drawing does not exist; cannot delete current drawing; cannot delete drawing (it is the work view); is only view in canned layout.

Current Drawing

Synopsis

&CURDRW

Description

Reads the name of the current drawing or retrieves a specified drawing, as follows.

To retrieve a drawing: &CURDRW = 'drawing name'

To return current drawing name: MYNAME = &CURDRW

Characteristics

Read/Write

String

30 Characters

Import an Existing Part

Synopsis

RPATT'filespec' [{,matrix|csys,scale}] [,LAYER]
[,PLMODS,value] [,NOVIEW] [,RETCAM,number]
[,IFERR,label:]

Description

Imports an existing part file from the specified directory into the currently displayed part. This statement is not object-valued and does not import the part as a group. For importing grouped parts, use the **RPATTG** statement.

Parameters

'filespec'

The filespec parameter contains the file specification data for the file to be imported.

matrix

A previously defined matrix which determines the location and orientation (or orientation and scale) of the part being imported.

csys,scale

A previously defined coordinate system and a scale factor which determine the orientation and scale of the part being imported. The scale factor must be a positive value.

LAYER

Minor word which causes the part to be imported onto the same layers as when it was created.

PLMODS

Minor word which indicates that a parts list parameter will be specified.

value

Integer between 1 and 4 inclusive which specifies how the system will handle parts list data associated with the part being imported. These values are as follows:

- Indicates that no parts list information will be imported. This is the default.

- Indicates that the default parts list format data will be imported. When this happens, the default data replaces the parts list format data of the current part.
- Indicates that the imported part will be added to the parts list as a single group (one-object entry).
- Indicates that all parts list entries on the imported part will be added individually to the parts list data of the current part. The parts list format data will also be imported.

NOVIEW

Minor word which indicates that no views are imported when the part is imported. The geometry is imported into the existing views based on the current model mode (**MODEL** or **VIEW DEPENDENT**). If **NOVIEW** is omitted, the views in the part being imported are also imported.

RETCAM

Minor word which controls how manufacturing data (tools, parameter sets, and operations) is handled.

number

Integer between 1 and 3 inclusive which specifies how the system will handle duplicate manufacturing object names in both the current and the imported part. These values are as follows:

- Indicates that no manufacturing data will be imported. This is the default.
- Indicates that the operation will terminate and no objects will be imported when duplicate manufacturing names occur.
- Indicates that the operation will continue and duplicate manufacturing objects will be renamed when duplicate manufacturing names occur.

IFERR,label:

The **IFERR** parameter specifies a label to which the program will jump if an error occurs.

NOTE

If a non-orthogonal matrix is used to transform a part containing arcs, conics, drafting objects or dimensions, the error message **CANNOT TRANSFORM** displays.



<i>Example</i>	This example demonstrates the use of the RPATT statement to import a previously created part into the current part. The part is imported through a matrix to place its origin at 1,1.
<i>Declarations</i>	NUMBER/M(12)
<i>Matrix Definition</i>	M = MATRIX/TRANSL,1,1,0
<i>Import Part</i>	RPATT/'STAR',M

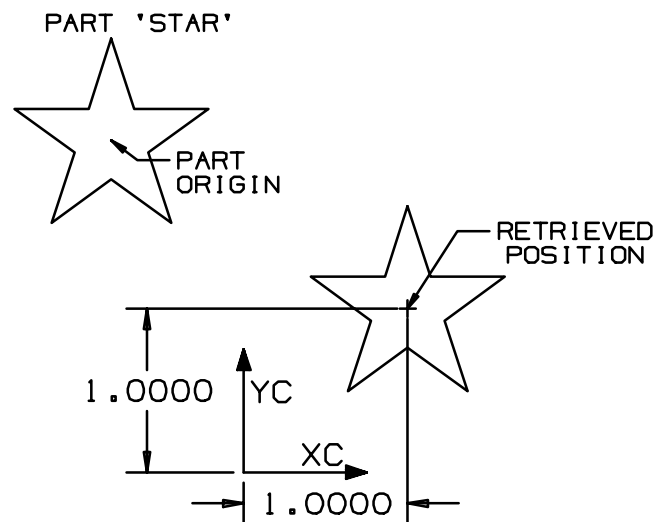


Figure 14–1 Part imported using the RPATT statement

Import an Existing Part Grouped

Synopsis **obj** = RPATTG/'filespec'[{,matrix | **csys,scale**}] [,LAYER] [,PLMODS,value] [,NOVIEW] [,RETCAM,number] [,IFERR,label:]

Description Imports an existing part file into the currently displayed part as a group. This statement is referred to as an object-valued function because the resulting group is a singular object which may be assigned to a previously declared object variable. In addition, the group automatically assumes the filename portion of the filespec parameter as its object name.

Parameters **'filespec'**
The filespec parameter contains the file specification data for the file to be imported.

matrix

A previously defined matrix which determines the location and orientation (or orientation and scale) of the part being imported.

csys,scale

A previously defined coordinate system and a scale factor which determine the orientation and scale of the part being imported. The scale factor must be a positive value.

LAYER

Minor word that causes the part to be imported onto the same layers as when it was created.

PLMODS

Minor word which indicates that a parts list parameter will be specified.

value

Integer between 1 and 4 inclusive which specifies how the system will handle parts list data associated with the part being imported. These values are as follows:

1. Indicated that no parts list information will be imported. This is the default.

2. Indicates that the default parts list format data will be imported. When this happens, the default data replaces the parts list format data of the current part.
3. Indicates that the imported part will be added to the parts list as a single group (one–object entry).
4. Indicates that all parts list entries on the imported part will be added individually to the parts list data of the current part. The parts list format data will also be imported.

NOVIEW

Minor word which indicates that no views are imported when the part is imported. The geometry is imported into the existing views based on the current model mode (**MODEL** or **VIEW DEPENDENT**). If **NOVIEW** is omitted, the views in the part being imported are also imported.

RETCAM

Minor word which controls how manufacturing data (tools, parameter sets, and operations) is handled.

number

Integer between 1 and 3 inclusive which specifies how the system will handle duplicate manufacturing object names in both the current and the imported part. These values are as follows:

- Indicates that no manufacturing data will be imported. This is the default.
- Indicates that the operation will terminate and no objects will be imported when duplicate manufacturing names occur.
- Indicates that the operation will continue and duplicate manufacturing objects will be renamed when duplicate manufacturing names occur.

IFERR,label:

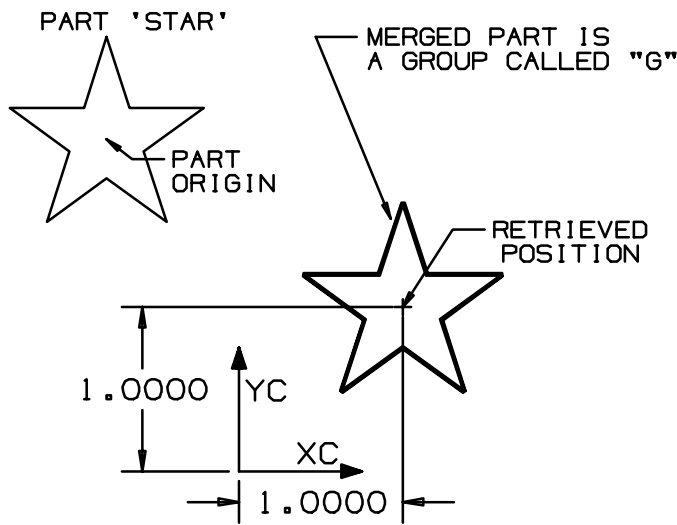
The **IFERR** parameter specifies a label to which the program will jump if an error occurs.

Example This example demonstrates the use of the **RPATTG** statement to import a previously created part into the current part as a group. The part is imported through a matrix to place its origin at 1,1.

Declarations ENTITY/G
NUMBER/M(12)

Matrix Definition M = MATRIX/TRANSL,1,1,0

Assemblies G=RPATTG/'STAR',M



NOTE The merged part is a group whose object attribute name is "STAR". The group object identifier is known by the variable named "G" in this program segment.

Figure 14–2 Part imported using the RPATTG statement

Activity: Drawing Creation

Write a program to allow the user to select a drawing size from a menu, create a drawing with the selected size, and import a drawing border.

- Use the CHOOSE/ statement to create the following menu:

Cue: Select Drawing Size

B-Size

C-Size

D-Size

- Use the DRAWC/ statement to create a drawing called "SHEET1". The size of the drawing will depend on the menu option selected.
- Use the RPATTG/ statement to import a border.
- Depending on the option selected, import one of the following parts.

grp_border_b.prt

grp_border_c.prt

grp_border_d.prt

(This Page Intentionally Left Blank)

Dimensions and Drafting Aids

Lesson 15

Objectives

- Gain a broad understanding of the commands available to add dimensions, notes, and labels.
- Recognize the GPA's and EDA's that are useful when working with dimensions and drafting aids.

Dimensions

There are statements available to help you with a number of dimensioning functions, including the creation of linear dimensions, angular dimensions, and diameter dimensions.

Many of the commands in this lesson contain arguments which can be expressed in several ways.

1. When you see the word “**(origin)**”, you may replace it with one of the following formats:

X,Y coordinates

The X and Y coordinates of a position in the current work plane.

```
NOTE/5,5,'THIS IS A NOTE'
```

Point

In a previously defined Point only the X and Y coordinate are used.

```
P1 = POINT/5,5  
NOTE/P1,'THIS IS A NOTE'
```

Delta Distance

The delta format positions the new text a delta distance from a previously defined point or from a text origin of a previously defined note or dimension.

```
P1 = POINT/5,5  
NOTE/DELTA,P1,2,1,'THIS IS A NOTE'
```

Delta Character Size

The delta character size positions the text to offset distance in multiples of the current character size from a previously defined point or origin of a drafting object.

```
P1 = POINT/5,5  
&CSIZE=.25  
NOTE/DELCH,P1,4,8,'THIS IS A NOTE'
```



2. When you see the argument “**arc position**”, replace it with one of the following words:

ENDOF This refers to an end point of the arc.

CENTER This refers to the center of the arc.

TANTO This refers to the tangency of the arc on the side specified by the positional modifier.

3. When you see the argument “**dim text**”, replace it with one of the following choices:

DIVIDE, number

This feature takes the dimension of the selected objects, divides it by the number entered and uses the result as text for the dimension.

'dimension text'

You will use this when you are showing dimensions which are different from the actual size of the drawing. The actual dimension is ignored and the data you enter is used for the dimension. The text field may contain a maximum of 132 characters. Although the maximum length for a single string is 78 characters long, you may concatenate strings by using the “+” sign.

4. When you see the argument “**app text**”, replace it with the following:

APPEND,text[,text]...

This feature allows you to append up to 200 characters of text to a dimension. When you separate two strings by a comma instead of a “+” sign, the string following the comma is put on a new line in the dimension.

5. When you see the words “**OBLIQ,angle**”, you may replace it with the following:

OBLIQ,angle

This option lets you angle extension lines as indicated. The lines will rotate around the end of the extension line which is attached to the object. A positive angle will result in a counterclockwise rotation of the extension lines. A negative angle will result in a clockwise rotation.

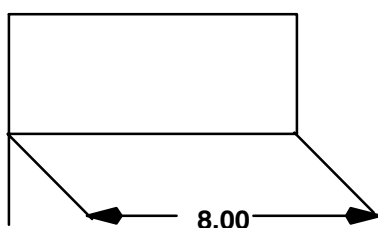


Figure 15–1 Dimension with Oblique Angular of 45 degrees

Creating Drafting Objects on Drawings

In GRIP, you can place dimensions and drafting aids on a drawing in either of two ways.

- Create the drafting objects in a view within the modeling display, and then move the view to the drawing. This causes the drafting objects to get moved to the drawing.
- Create the objects directly on the drawing itself.

Each of these methods is described below:

Creating Drafting Objects in a View

If a drawing is active, and you wish to create the drafting objects in a view (within the modeling display), follow the steps listed below:

1. Go to the modeling display using `&DSTATE=1`.
2. Create drafting objects in the modeling display.
3. Go back to the drawing using `&DSTATE=2`.
4. Place the view on the drawing using `DRAWE/ADD`. This causes the drafting objects to move to the drawing.

NOTE

Objects created on the drawing (e.g., curves) may be referenced directly on the drawing.

Creating Drafting Objects on the Drawing Itself

If a drawing is active, and you wish to create the drafting objects directly on the drawing itself, you must specify in what member view the object is to be referenced. This information must be input to the GRIP command, by adding the optional minor word `VIEW` with the view name after the object id in the GRIP statement. If the referenced object is on the drawing itself, then the drawing view name should be used. If a drawing is active and no view name is input, then the system defaults to the drawing view.

Note, that an error is reported if the view specified does not exist in the current drawing or if the object does not exist in that view (i.e. the object is view dependent in another view).

The following are some useful tips:

- When using screen positions (the POS/ command) to place dimensions on drawings, you must remember to map the position from the view to the drawing. See the MAP/ command for this information.
- You can use the &VWSEL GPA to get information on which view a selection was made or a position was specified.
- You should use &VWCURS to set the view of the cursor based on what operation the program is performing. The view should be = &ANY during object selection and = &WORK while specifying positions. These screen positions are to be used as object origins.

Table 15–1 Dimensioning Commands

Function	Format *
Horizontal and Vertical	LDIM/{HORIZ VERT},{origin}, [{ENDOF CENTER TANTO},] “P _{MOD3} ”,obj1, [VIEW,'View Name',] [{ENDOF CENTER TANTO},] “P _{MOD3} ”,obj2,[VIEW,'View Name'] [,Dim. text][,APPEND,App. text] [,OBLIQ, Angle]
Angular	ADIM/[MAJOR,]{origin},“P _{MOD3} ”, <u>line1</u> , [VIEW,'View Name',]“P _{MOD3} ”, <u>line2</u> [,VIEW,'View Name'] [,Dim. text][,APPEND,App. text]
Radius	RDIM/{origin},arc[,VIEW,'View Name'] [,Dim. text][,APPEND,App. text]

* (origin) is {X,Y or point or DELTA, {point or dimen},dx,dy}
 “P_{MOD2}” is {XSMALL or YSMALL or XLARGE or YLARGE}

Horizontal and Vertical Dimensions

Synopsis

obj = LDIM/{**HORIZ|VERT**},{**origin**},
 [{**ENDOF|CENTER|TANTO**},{**PMOD3**},{**obj1**,
 [**VIEW**,**'View Name'**],[{**ENDOF|CENTER|TANTO**},{
 "**PMOD3**",**obj2**[**VIEW**,**'View Name'**]
 [**Dim. text**],[**APPEND**,**App. text**],[**OBLIQ**,**Angle**]

Description

Creates a horizontal or vertical dimension based on which minor word is specified between two previously defined objects. Positional modifiers are used for each object to specify which end of the object should be dimensioned.

You may also override the dimension text by placing a string in the proper field. Text may be appended to the dimension. The placement of the appended text may be controlled by the global parameter **&APSITE**.

The extension lines of the dimension may be created at an angle measured by specifying an oblique angle.

Parameters

HORIZ

Minor word which indicates that the desired dimension is the horizontal distance between two objects.

VERT

Minor word which indicates that the desired dimension is the vertical distance between two objects.

(origin)

The dimensional text location. See the beginning of this section for the methods of specification.

ENDOF

Minor word which indicates that the extension line will terminate at one end of the arc if the specified object is an arc. The end to be considered will be further determined by a positional modifier (**PMOD3**).

CENTER

Minor word which indicates that the extension line will terminate at the center of the arc if the specified object is an arc. For this option the positional modifier (**PMOD3**) is omitted.

TANTO

Minor word which indicates that the extension line will terminate at one of the arc tangents if the specified object is an arc. The tangent to be considered will be further determined by a positional modifier (**PMOD3**).

PMOD3

Determines which end of the object or position on the arc is to be dimensioned. If you are dimensioning points or arc centers, you can omit the positional modifier. For all other objects, the positional modifier must be present.

obj1,obj2

The two existing objects between which the horizontal or vertical dimension will be created.

VIEW

Minor word that indicates that a view name is to follow.

'View Name'

This is a string variable or literal that is the view name.

Dim. text

A string of up to 50 characters you may enter if the system generated value and text are not desirable. You can enter the desired dimension and/or text between single quotes ('dimension text'), or you can use a previously declared string variable.

APPEND

Minor word which indicates that the text in the next field should be appended to the dimension text.

App text

Text string which will be appended to the dimension text (APPEND,'string','string'...). Each string constitutes a new line and may contain a maximum of 132 characters. The maximum number of characters which may be appended to a dimension is 600. The placement of appended text is controlled by the global parameter **&APSITE**.

OBLIQ

Minor word which indicates that the extension lines will be at a specified angle to their normal orientation.

Angle

The angle, in degrees, at which the extension lines will be rotated from their normal orientation. The rotation will be about the end of the line near the object. A positive angle generates a counterclockwise rotation, and a negative angle generates a clockwise rotation.

Example This example demonstrates the creation of a horizontal and a vertical dimension between previously defined points. The text DIM1 and DIM2 is appended to the dimension text.

Declarations ENTITY/PT0,PT1,PT2,DIM1,DIM2

Geometry Definition PT0 =POINT/0,0
PT1 =POINT/-1,0
PT2 =POINT/.5,-1

Dimension Definition DIM1=LDIM/HORIZ,-.5,1,PT1,PT0,APPEND,'DIM1'
DIM2=LDIM/VERT,1,-.5,PT0,PT2,APPEND,'DIM2'

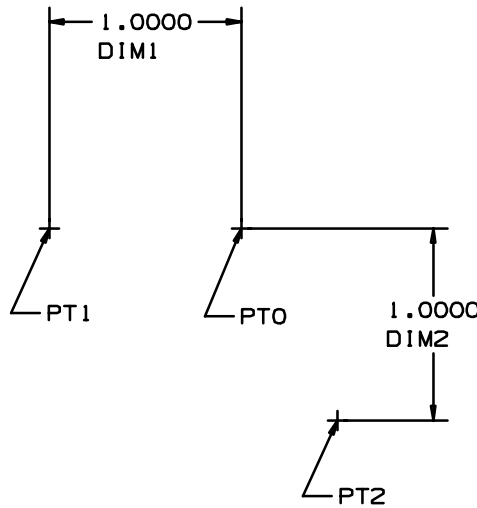


Figure 15–2 Horizontal and vertical dimensions

Angular Dimensions

Synopsis

obj = ADIM/[**MAJOR**,](**origin**),“**PMOD3**”,**line1**,
[**VIEW**,’**View Name**’,]“**PMOD3**”,**line2**
[,**VIEW**,’**View Name**’]
[,**Dim. text**][,**APPEND**,**App. text**]

Description

Creates an angular dimension between two previously defined lines. The minor angle is measured and displayed in the work coordinate system counterclockwise from the end of the first line to the end of the second line.

If you specify the minor word **MAJOR**, the opposite portion of the angle (360 minus minor) is dimensioned. Since the default is the minor portion, you do not need to use the minor word **MINOR**.

Parameters

MAJOR

An optional minor word which indicates that the major angle between the two lines is to be dimensioned. If omitted, the default is to the minor angle.

(origin)

The dimensional text location. See the beginning of this section for the methods of specification.

PMOD3

Determines which end of the line is to be dimensioned.

line1,line2

The two existing lines to be dimensioned.

VIEW

Minor word that indicates that a view name is to follow.

’View Name’

This is a string variable or literal that is the view name.

Dim. text

A string of up to 50 characters you may enter if the system generated value and text are not desirable. You can enter the desired dimension and/or text between single quotes (’dimension text’), or you can use a previously declared string variable.

APPEND

Minor word which indicates that the text in the next field should be appended to the dimension text.

App text

Text string which will be appended to the dimension text (APPEND,'string','string'...). Each string constitutes a new line and may contain a maximum of 132 characters. The maximum number of characters which may be appended to a dimension is 600. The placement of appended text is controlled by the global parameter **&APSITE**.

Example

This example demonstrates the creation of several angular dimensions between two previously defined lines. The objects used in the angular dimension must be lines. The major and minor dimensions have been created.

Declarations

ENTITY/LN1, LN2, DIM(4)

Geometry Definition

LN1 =LINE/0,0,.5,.5
LN2 =LINE/0,0,.5,-.5

Dimension Definition

DIM(1) = ADIM/.5,1.3,XLARGE, LN1, XLARGE, LN2, \$
APPEND, 'DIM(1)'
DIM(2) = ADIM/2.2,.21, XLARGE, LN1, XLARGE, LN2, \$
'MINOR', APPEND, 'ANGLE', 'DIM(2)'
DIM(3) = ADIM/MAJOR, -.05,.15, XLARGE, LN1, \$
XLARGE, LN2, APPEND, 'DIM(3)'
DIM(4) = ADIM/MAJOR, -1.8,.21, XLARGE, LN1, \$
XLARGE, LN2, 'MAJOR', APPEND, 'ANGLE', 'DIM(4)'

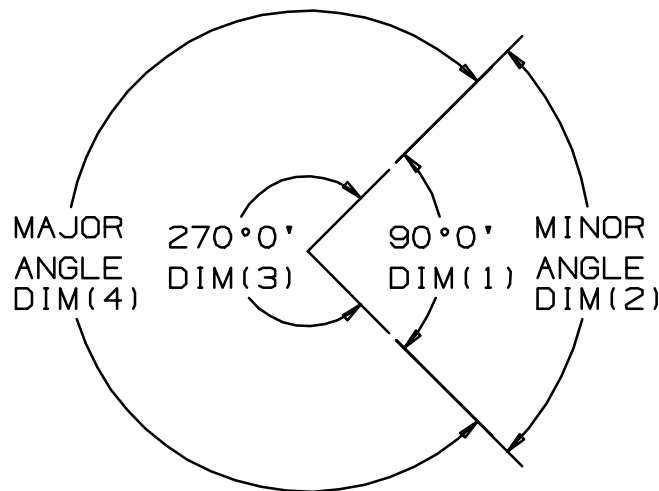


Figure 15–3 MAJOR and MINOR angular dimensions

Radius Dimensions

Synopsis

**obj = RDIM/(origin),arc[,VIEW,'View Name'][,Dim. text]
[,APPEND,App. text]**

Description

Creates a radius dimension by specifying a previously defined arc.

Parameters

(origin)

The dimensional text location. See the beginning of this section for the methods of specification.

arc

The existing arc to be dimensioned.

VIEW

Minor word that indicates that a view name is to follow.

'View Name'

This is a string variable or literal that is the view name.

Dim. text

A string of up to 50 characters you may enter if the system generated value and text are not desirable. You can enter the desired dimension and/or text between single quotes ('dimension text'), or you can use a previously declared string variable.

APPEND

Minor word which indicates that the text in the next field should be appended to the dimension text.

App text

Text string which will be appended to the dimension text (APPEND,'string','string'...). Each string constitutes a new line and may contain a maximum of 132 characters. The maximum number of characters which may be appended to a dimension is 600. The placement of appended text is controlled by the global parameter **&APSITE**.

Example This example demonstrates the creation of the radius dimension of a previously defined arc.

Declarations ENTITY/CR1,DIM1

Geometry Definition CR1=CIRCLE/0,0,1,START,0,END,270
&ENSITE=&TOPR

Dimension Definition DIM1 =RDIM/-1.1,.75,CR1,APPEND,'DIM1'

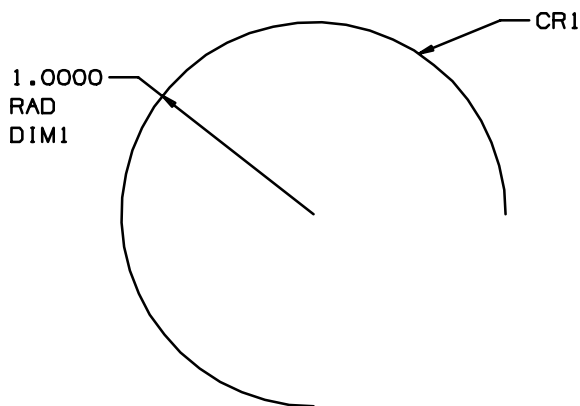


Figure 15–4 Arc radius dimension

Dimension Statements

The following pages provide detailed information on the GRIP statements used to create the objects which are defined in Unigraphics using the Dimension Creation option. The statements are grouped to mimic Unigraphics to help you find the specific statement you want as quickly as possible.

Since all of the dimensioning statements require you to specify an origin parameter and since the origin parameter has several options, this parameter will be covered now, collectively, rather than with each individual statement.

The GRIP words listed below are described on the following pages. The listing below reflects the order in which these words appear in the following pages, not alphabetical order.

GPAs and Constants	Major Words	Minor Words
&TXARR	LDIM	HORIZ
&AUTO	ADIM	VERT
&ARRIN	RDIM	ENDOF
&ARROUT		CENTER
&ENSITE		TANTO
&TOPL		OBLIQ
&TOPC		MAJOR
&TOPR		APPEND
&MIDL		
&MIDC		
&MIDR		
&BOTL		
&BOTC		
&BOTR		
&XLINE		
&BOTH		
&FIRST		
&SECOND		
&NONE		
&APSITE		
&LEFT		
&RIGHT		

The GRIP drafting and dimensioning functions allow you to create drafting objects (such as notes and labels), create dimensions, and control the drafting and dimension parameters. The following tables list and define drafting parameters, dimensioning commands, and drafting symbols which are presented in this lesson.

Table 15–2 Drafting Parameters

Parameter	GPs and Constants			
Text and arrow location	&TXARR	RW	N	[1..3]
Automatic	&AUTO	C	N	=1
Manual loc., arrows in	&ARRIN	C	N	=2
Manual loc., arrows out	&ARROUT	C	N	=3
Character size	&CSIZE	RW	N	GT 0
Places past the decimal for numerical displays	&DECPL	RW	N	[0..9]
Dimension places past the decimal	&DDECPL	RW	N	[0..7]
Fraction display type	&FTYPE	RW	N	[1..4]
Decimal	&DECIM	C	N	=1
2/3 size common	&FRAC	C	N	=2
3/4 size common	&TQFRAC	C	N	=3
Full size common	&FSFRAC	C	N	=4
Extension line display	&XLINE	RW	N	[1..4]
Both lines	&BOTH	C	N	=1
First line only	&FIRST	C	N	=2
Second line only	&SECOND	C	N	=3
No ext. lines	&NONE	C	N	=4
Entity site related to text	&ENSITE	RW	N	[1..9]
Top Left	&TOPL	C	N	=1
Top Center	&TOPC	C	N	=2
Top Right	&TOPR	C	N	=3
Mid–left	&MIDL	C	N	=4
Mid–center	&MIDC	C	N	=5
Mid–right	&MIDR	C	N	=6
Bottom Left	&BOTL	C	N	=7
Bottom Center	&BOTC	C	N	=8
Bottom Right	&BOTR	C	N	=9



Name **&CSIZE** Character Size

Synopsis **&CSIZE**

Description Controls the size of all text characters.

There are individual text preferences for dimension main text, appended text, tolerance text and drafting aid text. The Write option for this GPA sets all of these preferences. The Read option can only return one of these preferences. It returns the last preference that was modified. For example, if the DIMPAR command is used to set the preferences from a Note, then the GPA returns the text preferences for drafting aid text. If the DIMPAR command is used to set the preferences from a dimension, then the GPA returns the text preferences for dimension main text.

Characteristics Read/Write Number Greater Than Zero

Extension Line Display

Synopsis

&XLINE

Description

Controls the display of the extension lines for all dimensions.

Characteristics

Read/Write Number [1..4]

Constant Values

1 = &BOTH
2 = &FIRST
3 = &SECOND
4 = &NONE

Parameters

&BOTH

GPA which indicates that both extension lines will be displayed.

&FIRST

GPA which indicates that only the first extension line will be displayed.

&SECOND

GPA which indicates that only the second extension line will be displayed.

&NONE

GPA which indicates that neither extension line will be displayed.

Text and Arrow Location

Synopsis

&TXARR

Description

Controls the text and arrow location parameter. In Unigraphics, text and arrow location also allows you to enter dimension text manually. Entering dimension text manually in GRIP is controlled in the dimension statements themselves, so this GPA controls only text and arrow location.

Characteristics

Read/Write Number [1..3]

Constant Values

1 = &AUTO
2 = &ARRIN
3 = &ARROUT

Parameters

&AUTO

GPA which indicates that the text and arrows will be placed automatically, as they fit between the extension lines, in the following order:

Text centered between the extension lines with the arrows inside.

Text centered between the extension lines with the arrows outside.

Text placed outside the extension lines with the arrows inside.

Text and arrows placed outside the extension lines.

&ARRIN

GPA which indicates that the text will not be automatically centered between the extension lines. Its location is determined by the origin parameter in the dimension statement. The arrows are placed inside the extension lines.

&ARROUT

GPA which indicates that the text will not be automatically centered between the extension lines. Its location is determined by the origin parameter in the Dimension Statement. The arrows are placed outside the extension lines.

Entity Site related to Text

<i>Synopsis</i>	&ENSITE
<i>Description</i>	Controls the location of the point used to place a drafting object in relation to its text.
<i>Characteristics</i>	Read/Write Number [1..9]
<i>Constant Values</i>	1 = &TOPL 2 = &TOPC 3 = &TOPR 4 = &MIDL 5 = &MIDC 6 = &MIDR 7 = &BOTL 8 = &BOTC 9 = &BOTR
<i>Parameters</i>	&TOPL GPA which indicates that the point used to locate text will be at the top left corner of an imaginary box containing the text. &TOPC GPA which indicates that the point used to locate text will be at the center of the top edge of an imaginary box containing the text. &TOPR GPA which indicates that the point used to locate text will be at the top right corner of an imaginary box containing the text. &MIDL GPA which indicates that the point used to locate text will be at the mid-point of the left edge of an imaginary box containing the text. &MIDC GPA which indicates that the point used to locate text will be in the center of an imaginary box containing the text.



&MIDR

GPA which indicates that the point used to locate text will be at the mid-point of the right edge of an imaginary box containing the text.

&BOTL

GPA which indicates that the point used to locate text will be at the bottom left corner of an imaginary box containing the text.

&BOTC

GPA which indicates that the point used to locate text will be at the center of the bottom edge of an imaginary box containing the text.

&BOTR

GPA which indicates that the point used to locate text will be at the bottom right corner of an imaginary box containing the text.

Drafting Aids

The GRIP statements, which are described in this lesson, are used to create the objects which are defined in Unigraphics using the Drafting Aids option.

There are two sections in this lesson which are not based on a specific GRIP statement. These are text control, and drafting and dimensioning symbols which deal with the manipulation of text using special control characters.

Table 15–3 Dimensioning Commands

Function	Format *
Note	NOTE/(origin),{scratch file #1 [,IFERR,label:]} 'text'[, 'text']+
Label	LABEL/[{LEFT RIGHT},](origin), {obj[,VIEW,'View Name'] [, (origin)] (origin)} 'text'[, 'text']+
Linear Centerline	CLINE/LINEAR,obj_list[,VIEW,view name list]

Note

Synopsis

obj = NOTE/(origin),
 {scratch file #1[,IFERR,label:] |'text'[, 'text'] + }

Description

Creates a note by specifying the text origin and either a text string or scratch file 1. If you specify scratch file 1, the statement will read whatever text it finds into the note.

You may also specify and **IFERR** label to jump to, if the **NOTE** statement cannot read scratch file 1.

Parameters

(origin)

The text location. See the beginning of this section for the methods of specification.

scratch file #1

The text which exists in scratch file number one may be used for the note. Each line of text in this file will be on a separate line in the note.

IFERR,label:

If scratch file number one is used for the text, the optional **IFERR** parameters may be used. This feature provides the capability of branching to another statement, specified by (**LABEL:**), in the program if an error occurs in reading the scratch file.

'text','text'

The 'text' parameters may be either text between single quotes or string variables. In either case, each text parameter is placed on a new line. As many lines as desired may be included in the note when using a scratch file. Because of downstream application limits, notes should not exceed 132 characters per line or more than 600 characters per note. This includes special control characters used to generate GD&T symbols, diameter symbols, etc. If you create text with these symbols, you must include the special control characters in the total character count..

Example

This example demonstrates the creation of a note by specifying a text string and by reading scratch file 1.

NOTE

As input to a note statement, the following text will be written to a scratch file:

1. MATERIAL: STEEL
2. REMOVE ALL BURRS AND SHARP EDGES.

Create and Write to Text File

```
CREATE/TXT,1,'NOTEDATA.TXT'  
WRITE/1,'2. REMOVE ALL BURRS AND'  
WRITE/1,' SHARP EDGES'
```

Note Definitions

```
NOTE/-2.2,1.5,'1. MATERIAL: STEEL.'  
NOTE/-1.9,0,1,IFERR,A1:  
JUMP/A2:  
A1:  
MESSG/TEMP,'ERROR READING FILE'  
A2:  
HALT
```

1. MATERIAL : STEEL .
2. REMOVE ALL BURRS AND SHARP EDGES .

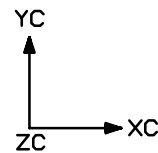


Figure 15–5 A programmed note and a note from scratch file # 1

Label

Synopsis

obj = LABEL/[{LEFT|RIGHT},](origin),
 {obj[,VIEW,'View Name'][, (origin)] |
 (origin)} 'text'[, 'text'] +

Description

Creates a label by specifying a text string and either an object or an origin where the arrow terminates. The **LABEL** statement can also be used to return an object.

Parameters

LEFT and RIGHT

Determines which side of the text the leader line will begin on, as viewed in a readable position. This parameter can also be controlled by using Global Parameters **&LEFT** and **&RIGHT** by entering the following statement:

&LEADER=&RIGHT or **&LEADER=&LEFT**

(origin)

The text location. See the beginning of this section for the methods of specification.

obj

An existing object at which the leader line may terminate. The object must be a point, line, arc, conic, or spline.

NOTE

Labels are associative objects; therefore, if the object specified for the leader line termination is deleted, both objects will be deleted.

VIEW

Minor word that indicates that a view name is to follow.

'View Name'

This is a string variable or literal that is the view name.

(origin)

If an object is specified for the leader line, the end position of the leader may be controlled by specifying a position, nearest the point on the object, where the leader line will terminate. If a position is not specified, the leader line will terminate at a point on the object nearest the origin of the text. See the beginning of this section for the methods of specification.

(origin)

If an object is not specified for the leader line, a position must be specified at which the leader line will terminate. See the beginning of this section for the methods of specification.

'text'[, 'text']+

The 'text' parameters may be either text between single quotes or string variables. In either case, each text parameter is placed on a new line. As many lines as desired may be included in the label with the restriction that no more than 600 characters can be contained in one label.

Example This example demonstrates the creation of two labels by specifying a text string and a previously created circle where the arrow will terminate.

Declarations ENTITY/CR1,LBL1,LBL2

Geometry Definition CR1 =CIRCLE/0,0,.5
&ENSITE=&TOPL

Label Definition LBL1 =LABEL/RIGHT,-2.2,1.5,CR1,'CIRCLE CR1'
LBL2 =LABEL/LEFT,.5,-1,CR1,0,-1,'THIS IS',
'CIRCLE CR1'
HALT

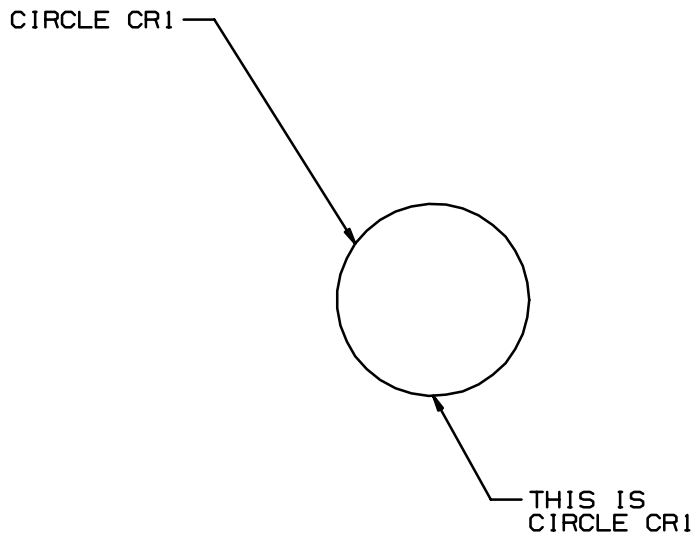


Figure 15–6 Labels

Linear Centerline

Synopsis

obj = CLINE/LINEAR,obj list[,VIEW,view name list]

Description

Creates a linear centerline object. The linear centerline can be created through a set of points or arcs or a single point or arc and lies in the X-Y plane of the work coordinate system.

Parameters

LINEAR

Minor word that causes a linear centerline to be generated.

obj list

A list of either points or arcs. The object list must contain at least one object. It may contain up to 100 objects.

NOTE

The objects must be colinear, if they are not colinear, you will get a run time error.

Linear centerlines are associative; therefore, if the generating objects are deleted, the centerline will be deleted.

VIEW

Minor word that indicates that a view name is to follow.

view name list

A list of strings which represents the view names. If you specify only a single view name for a view name list that corresponds to an object list, then the single view name is used for all the objects in the corresponding object list.

Example

This example demonstrates the creation of two linear centerlines using colinear points and circles.

```

$$
$$ Declarations
$$

ENTITY/P(4),CR(9),CTRLN(4)

$$
$$ Geometry Definitions
$$

P(1)=POINT/-1.5,1.25
P(2)=POINT/-.5,1.25
P(3)=POINT/.5,1.25
P(4)=POINT/1.5,1.25
CR(1)=CIRCLE/-1.5,.5,.125
CR(2)=CIRCLE/-.5,.5,.125
CR(3)=CIRCLE/.5,.5,.125
CR(4)=CIRCLE/1.5,.5,.125

$$
$$ Centerline Definition
$$

CTRLN(1)=CLINE/LINEAR,P(1),P(2),P(3),P(4)
CTRLN(2)=CLINE/LINEAR,CR(1),CR(2),CR(3),CR(4)

$$
$$ Program continues
$$

.
.
.

```

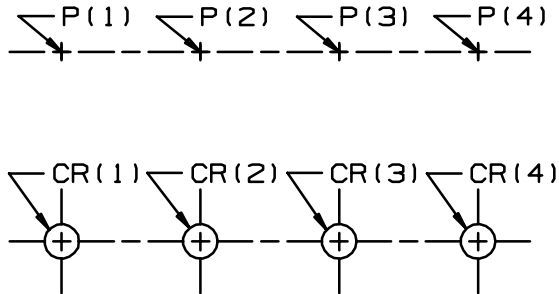


Figure 15-7 Example of CLINE statement used for linear centerline

Drafting EDAs: Drafting Object Origin*Synopsis***&ORIGIN(obj)***Description*

Returns the location in X, Y, and Z in the current work coordinate system of the origin of a specified drafting object to a receiving numerical array. The origin of a drafting object can also be altered by writing coordinates to this EDA.

Characteristics

Read/Write Three Position Numerical Array

 \pm Infinity*Parameters***obj**

Existing drafting object whose origin will be accessed.

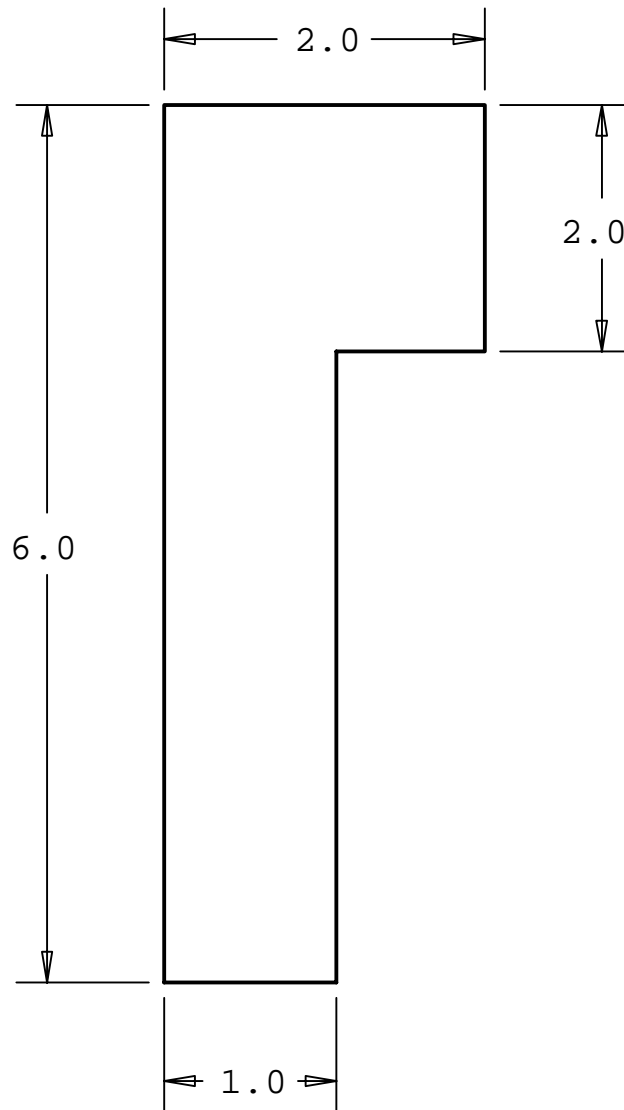
Drafting EDAs: Text

<i>Synopsis</i>	&DMTEXT(obj)
<i>Description</i>	<p>Returns the text of a specified dimension or drafting aid to a receiving variable or GRIP statement.</p> <p>If the note contains more than one line of text, each line can be returned to a properly declared string array. The first line takes the first array position, the second line, the second position, etc.</p>
<i>Characteristics</i>	Read Only String 132 Characters Per Line
<i>Parameters</i>	<p>obj Existing drafting object whose text will be returned.</p>
<i>Example</i>	This example demonstrates the use of the &DMTEXT EDA to write a note using text from a dimension.
<i>Declarations</i>	<pre> ENTITY/L(4),DIM1 STRING/STR(30) BCK010: PARAM/'ENTER STK DIMS', \$ 'LENGTH',LNG,\$ 'DIAMETER',DIA,RESP JUMP/L10:;,TERM:;,RESP L(1)=LINE/0,0,LNG,0 L(2)=LINE/LNG,0,LNG,DIA L(3)=LINE/LNG,DIA,0,LNG L(4)=LINE/0,LNG,0,0 DIM1=LDIM/HORIZ,LNG/2,3,XSMALL,L(3), \$ XLARGE,L(3) CYLDIM/LNG+1,ANG/2,YLARGE,L(2),YSMALL,L(2) STR=&DMTEXT(DIM1) NOTE/LNG/2,-.5,'REQUIRED MATERIAL STOCK = '+STR CANCEL:HALT </pre> <p>The note will be written with the same text as DIM1. There are other ways to create this note; however, this method assures that the decimal places in the note match the dimension.</p>



Activity: Dimensioning the L-shape

- Use the program created in the L-Shape activity. Add the dimensions shown in the original figure. The dimensions should be the last action of the program.

**Figure 15–8** Dimensioned L–Shape

(This Page Intentionally Left Blank)

Attributes (Optional)

Lesson 16



Objectives

- Learn the GRIP statements used to assign, use and delete attributes.

Many of the interactive attribute options, especially **NAME**, are performed in GRIP using GPAs and EDAs. See the respective chapters on GPAs and EDAs found in this manual.

Object Names

Using objects in a GRIP program, you have learned how to use an **ENTITY/** declaration statement to define a program variable name and reserve space in the memory to store object identifiers. An object identifier is a system defined tag that is used to distinguish various types of data in a part data model.

The following table describes the EDAs and major words used with object attribute names. Notice that the **&ENAME** EDA requires a number (n) which indicates which occurrence of the object is returned. This means that object names are not unique, and there can be many objects with the same name in a GRIP program. The major word **ENUM/** will indicate the total number of objects with a specified name. Please note that you cannot set an object name to **&NULSTR**; you must use the **DELNAM/** major word for this.

Table 16–1 Object Name Commands

Type	Format
Read/set object name	&NAME({ <u>obj</u> 'name'})
Get number of named object	ENUM/'name'
Get nth object with specified name	&ENAME(n,'name'[IFERR, label:])
Delete Name	DELNAM/{ <u>obj list</u> ALL}
Assign Object Attributes	ASATT/{ <u>obj list</u> ALL PART 'name'},attributes list[,data_type]
Get Attribute Title	&ATTTL({ <u>obj</u> PART 'name'},seqno[,data_type])
Get Attribute Value	&ATTVL({ <u>obj</u> PART 'name'},'title'[IFERR,label:] [,data_type])
Delete Object Attribute	DLATT/{ <u>obj list</u> ALL PART 'name'},{title list ALL}[,data_type]
Control Name Display Location (X,Y,Z)	&ATDISL(<u>obj</u>)

&NAME

<i>Name</i>	&NAME	Object Name
<i>Synopsis</i>	&NAME({<u>obj</u> 'name'})	
<i>Description</i>	Assigns a name to an object as an attribute. The assigned name may be assigned to any number of objects. It is in addition to the geometric object name (LN1=LINE/X,Y,Z) which is declared and must be unique.	
<i>Characteristics</i>	Read/Write	String 30 Characters
<i>Parameters</i>	<p><u>obj</u> Existing object to which the name will be assigned or from which the name will be returned.</p> <p>'name' A string or string variable which represents the name of an object such as a reference set to which the name will be assigned or from which the name will be returned.</p>	
<i>Valid Characters</i>	Use upper case A - Z, digits 0 - 9, and the following special characters: "." period, "-" hyphen, "#" pound sign, "/" slash, and "_" underscore. Blank spaces may not be used. The maximum name length is 30 characters.	

Example This example demonstrates the use of the **&NAME** EDA to assign a name to several objects.

Declarations

```

ENTITY/LN1,LN2,LN3,LN4,C(4)
LN1=LINE/0,0,4,0
LN2=LINE/0,3,4,3
LN3=LINE/0,0,0,3
LN4=LINE/4,0,4,3
C(1)=CIRCLE/.5,.5,.25
C(2)=CIRCLE/3.5,.5,.25
C(3)=CIRCLE/.5,2.5,.25
C(4)=CIRCLE/3.25,2.5,.25
&ATTDIS=&ON
&NAME(LN1)='PART_EDGE'
&NAME(LN2)='PART_EDGE'
&NAME(LN3)='PART_EDGE'
&NAME(LN4)='PART_EDGE'
&NAME(C(1))='MOUNTING_HOLES'
&NAME(C(2))='MOUNTING_HOLES'
&NAME(C(3))='MOUNTING_HOLES'
&NAME(C(4))='OFFSET_HOLE'
    
```

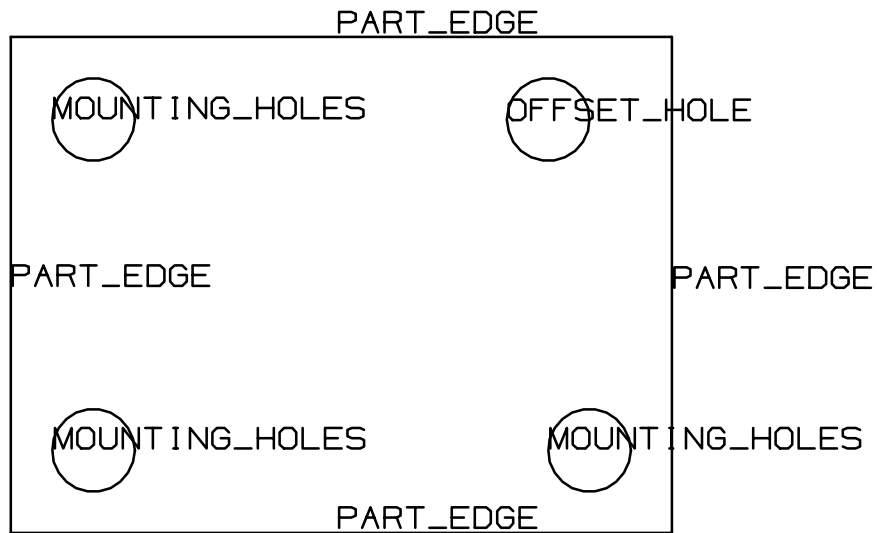


Figure 16–1 Assigning names to objects with attribute display on

ENUM

<i>Name</i>	ENUM	Get number of Named objects
<i>Synopsis</i>	num = ENUM/'name'	
<i>Description</i>	Returns the number of objects which have been assigned the same name by the EDA &NAME .	
<i>Parameters</i>	'name'	A literal string which is the name assigned to an object using the EDA &NAME .
<i>Example</i>	This example demonstrates the use of the ENUM statement to verify the occurrences of the &NAME , PART_EDGE.	
<i>Declarations</i>	<pre> ENTITY/LN(4) LN(1)=LINE/-1,-1,1,-1 LN(2)=LINE/1,-1,1,1 LN(3)=LINE/1,1,-1,1 LN(4)=LINE/-1,1,-1,-1 &NAME(LN(1))='PART_EDGE' &NAME(LN(2))='PART_EDGE' &NAME(LN(3))='PART_EDGE' &NAME(LN(4))='PART_EDGE' ENTNUM=ENUM/'PART_EDGE' </pre>	
	NOTE	The numerically valued variable ENTNUM is assigned the number 4.

&ENAME

Name **&ENAME** Find the Nth Object with Given Name

Synopsis **&ENAME(n,'name'[,IFERR,label:])**

Description Returns the object identifier of the nth object with the specified name to a receiving variable or GRIP statement. You can use this function to identify objects which were named interactively or in another GRIP program. This EDA does not guarantee the order in which objects are returned. Since object identifiers may change across sessions, &ENAME returns object identifiers without regard to a particular order.

NOTE The following object types are not valid for this EDA: 60, 61, 62, and 64 which correspond to View, Layout, Drawing, and Reference Set.

Characteristics Read Only Object 30 Characters

Parameters **n**
An integer representing the nth object in the data base with the given name.

'name'
The name, which can have a maximum of 30 characters, for which the system will search.

NOTE You can use an asterisk character (*) as a wild card.

IFERR,label:

Specifies a label to which the program jumps if an error occurs. An error occurs if there are no objects with the specified name or if the number of objects with the given name is less than the value specified by "n".

Example This example demonstrates the use of the **&ENAME** EDA to find the third object with the name "SPOOL5" in order to change it to "SPOOL6".

Declarations ENTITY/E
E=&ENAME(3,'SPOOL5',IFERR,A1:)
&NAME(E)='SPOOL6'
A1:MESSG/'CANNOT FIND NAME'

In the next program, the first ten objects or the total number of objects, whichever is smaller, with the name LOCK_PIN will be deleted.

Declarations

```
ENTITY/E(10)
DO/A1:,I,1,10
E(I)=&ENAME(I,'LOCK PIN',IFERR,A2:)
A1:
A2:
I=I-1
IF/I=0,JUMP/A3:
DELETE/E(1..I)
A3:

XYZ(1)=.5
XYZ(2)=.5
&ATDISL(LN1)=XYZ
```

DELNAM

Name **DELNAM** Delete Name

Synopsis **DELNAM/{obj list|ALL}**

Description Permanently deletes the name assigned to an object.

Parameters **obj list**

A list of existing objects whose names you wish to delete.

ALL

Minor word which indicates that all objects in a given part which satisfy the following conditions are to have their names deleted:

1. The object must be a selectable type/subtype.
2. The object must not be blanked.
3. The object must reside on a layer which is selectable in the work view.

Example This example demonstrates the use of the **DELNAM** statement.

```
ENTITY/OBJ(2)
OBJ(1) = LINE/0,0,1,1
OBJ(2) = POINT/2,2,2

&NAME(OBJ(1)) = 'LINE1'
&NAME(OBJ(2)) = 'POINT1'

DELNAM/OBJ
HALT
```

ASATT

<i>Name</i>	ASATT Assign Object Attributes
<i>Synopsis</i>	ASATT/{<u>obj list</u> ALL PART 'name'}, attribute list[, data_type]
<i>Description</i>	<p>Allows you to assign object attributes to previously defined objects.</p> <p>NOTE GRIP forces the case of attribute titles to uppercase. Interactive Unigraphics only allows you to create and read uppercase attribute titles. Variable length strings can be upper or lowercase.</p>
<i>Parameters</i>	<p><u>obj list</u> The existing objects to which the attributes are to be assigned. If a PART object identifier is included, it must be the first member of the list.</p> <p>ALL Minor word which indicates that the attributes are to be assigned to every object in the part (except manufacturing objects).</p> <p>PART Minor Word which indicates that the attributes are to be assigned to the part instead of to individual objects.</p> <p>'name' A string or string variable which represents the name (object identifier) of an object such as a reference set.</p> <p>attribute list The attribute list is a string list which consists of an alternating series of titles and values. The list must consist of an even number of elements, where the odd numbered elements, 1, 3, 5, etc. are titles and the even numbered elements, 2, 4, 6, etc. are values. The string array length must be equal to the longest title and/or value in the list. However, the maximum length of a title is 50 characters and the maximum length of a value is 132 characters. The title cannot begin with a dollar sign character (\$).</p> <p>Restrictions:</p> <ul style="list-style-type: none"> • Integer Range is any valid Unigraphics number. Example: "13579" • Real has 8 characters including \pm, E sign. Example: "3.5 E 10" • Date and Time must be in the format "MM/DD/YY HH:MM" Example: "09/30/91 04:35"



The reference data type allows you to embed an expression in the reference string that uses a conversion specifier with the following syntax:

<Xm.n@exp_name> or <Xm,n@exp_name> where:

- <> the enclosing angle brackets identifies the conversion specifiers and expression name for the reference.
- X indicates a reference to an expression.
- m.n is the conversion specification for the f format used in the C language. “m” specifies the minimum field width. If necessary, it is padded on the left to make up the field width. “n” specifies the maximum number of digits after the decimal point of the expression value. You can use a comma as the decimal point character by specifying “m,n” instead of “m.n”.
- @ indicates that an expression name is to follow.
- exp_name specifies the expression name on the left hand side of an expression.

[,data_type]

The parameter data_type has been assigned a value of the GPA &ATTTYPE. Depending on the value of data_type, even numbered elements in the attribute list are converted to the appropriate data type.

NOTE Attribute values are represented as strings, but they are internally converted to the specified data type. If the data type is not specified, the default data type is variable length string.

Part attributes can only have a data_type of 5 (string); the system ignores all other data_types for part attributes.

Data_Type	Explanation
1	Integer
2	Floating point
3	Date and time
4	Null
5	Variable length character
7	Reference

STRING/ATR(2)
 ATR(1)='TITLE'
 ASATT/ENT,ATR,3

ATR(2)=10
 ASATT/ENT,ATR1,1

Example

This example demonstrates the use of the **ASATT** statement.

Declarations

ENTITY/LN(4),CR(2)
 STRING/ATT(6,132)

Geometry Definition

LN(1) =LINE/-1,-1,1,-1
 LN(2) =LINE/1,-1,1,1
 LN(3) =LINE/1,1,-1,1
 LN(4) =LINE/-1,1,-1,-1
 CR(1) =CIRCLE/-.5,-.5,.25
 CR(2) =CIRCLE/.5,.5,.25
 ATT(1)='USE'
 ATT(2)='PART EDGE'

Attribute Assignment

ASATT/LN,ATT(1..2)

Reassign Value for

ATT(2)='MOUNTING HOLE'
 ATT(3)='DRILL'
 ATT(4)='.5 DIA. .75 DEEP'
 ATT(5)='TAP'
 ATT(6)='9/16 18 THREAD'

Attribute Assignment

ASATT/CR,ATT

This example would assign attributes to each object in the arrays LN and CR in the following manner:

Objects	Attribute Title	Attribute Value
LN(1..4)	USE	PART EDGE
CR(1..2)	USE	MOUNTING HOLE
CR(1..2)	DRILL	.5 DIA. .75 DEEP
CR(1..2)	TAP	9/16-18 THREAD

&ATTTL

Name **&ATTTL** Get Attribute Title

Synopsis **&ATTTL({obj|PART|'name'},seqno[,data_type])**

Description Returns the title of the attribute which is referred to by the specified sequence number.

Characteristics Read Only String 50 Characters

Parameters **obj**
Existing object from which attribute titles will be accessed.

PART

Minor word which indicates that the part attributes will be accessed.

'name'

A string or string variable which represents the name of an object such as a reference set.

seqno

Sequence number of the desired attribute title. The attribute titles are sequenced in a linear manner (e.g., the first title assigned is 1, the second title assigned is 2, and so on where each title is of a given data type).

[,data_type]

The parameter data_type has been assigned a value of GPA &ATTTYPE. If data_type is not specified, the default data type is variable length string.

NOTE Reference attributes do not use sequence numbers. Therefore, ATTTL does not return reference titles. Please refer to the notes on reference attributes in the attribute section of the *Unigraphics Essentials User Guide* for a description of the use of reference attributes.

Part attributes can only have a data_type of 5 (string); the system ignores all other data_types for part attributes.

Data Type	Description
1	Integer
2	Floating point
3	Date and time
4	Null string
5	Variable length character

Example

This example demonstrates the use of the **&ATTTL** EDA to extract an attribute title.

Declarations

```

$$ DECLARATIONS
  GRIPSW / DECLRV
  ENTITY / OBJ
  STRING / MYATTS(14,132), MYTTL(7,132), MYVAL(7,132)
  NUMBER / INDX

$$ CREATE OBJECT
  OBJ = CIRCLE / 1,1,1

$$ INITIALIZE AN INTEGER ATTRIBUTE
  MYATTS(1) = 'NUMBER OF FLOATS AND STRINGS'
  MYATTS(2) = '2'

$$ INITIALIZE FLOATING POINT ATTRIBUTES
  MYATTS(3) = 'HOLE DEPTH'
  MYATTS(4) = '1.0'
  MYATTS(5) = 'HOLE WIDTH'
  MYATTS(6) = '1.5'

$$ INITIALIZE A DATE AND TIME ATTRIBUTE
  MYATTS(7) = 'DATE CREATED'
  MYATTS(8) = '09/16/96 14:35'

$$ INITIALIZE A NULL ATTRIBUTE
  MYATTS(9) = 'NULL DATA'
  $$ MYATTS(10) = &NULSTR

$$ INITIALIZE STRING ATTRIBUTES
  MYATTS(11) = 'HOLE DESCRIPTION #1'
  MYATTS(12) = 'This circle represents a hole.'
  MYATTS(13) = 'HOLE DESCRIPTION #2'
  MYATTS(14) = 'The hole specifications are provided as

```

attributes.'

\$\$ ASSIGN THE ATTRIBUTES

```
ASATT / OBJ, MYATTS(1..2), 1
ASATT / OBJ, MYATTS(3..6), 2
ASATT / OBJ, MYATTS(7..8), 3
ASATT / OBJ, MYATTS(9..10), 4
ASATT / OBJ, MYATTS(11..12), 5
ASATT / OBJ, MYATTS(13..14)
```

\$\$ READ THE ATTRIBUTE TITLES AND VALUES

```
MYTTL(1) = &ATTTL(OBJ,1,1)
MYVAL(1) = &ATTVL(OBJ,MYTTL(1),1)
```

```
MYTTL(2) = &ATTTL(OBJ,1,2)
MYVAL(2) = &ATTVL(OBJ,MYTTL(2),2)
```

```
MYTTL(3) = &ATTTL(OBJ,2,2)
MYVAL(3) = &ATTVL(OBJ,MYTTL(3),2)
```

```
MYTTL(4) = &ATTTL(OBJ,1,3)
MYVAL(4) = &ATTVL(OBJ,MYTTL(4),3)
```

```
MYTTL(5) = &ATTTL(OBJ,1,4)
MYVAL(5) = &ATTVL(OBJ,MYTTL(5),4)
```

```
MYTTL(6) = &ATTTL(OBJ,1,5)
MYVAL(6) = &ATTVL(OBJ,MYTTL(6),5)
```

```
MYTTL(7) = &ATTTL(OBJ,2)
MYVAL(7) = &ATTVL(OBJ,MYTTL(7))
```

\$\$ OUTPUT THE REPORT

```
EJECT / PRINT
DO / DOLUP:, INDX, 1, 7
DOLUP: PRINT / MYTTL(INDX) + ' = ' + MYVAL(INDX)
```

HALT

The CRT would appear as follows:

```
NUMBER OF FLOATS AND STRINGS = 2
HOLE DEPTH = 1.000000
HOLE WIDTH = 1.500000
DATE CREATED = 9/16/96 14:35
NULL DATA =
HOLE DESCRIPTION #1 = This circle represents a hole.
```


HOLE DESCRIPTION #2 = The hole specifications are provided as attributes.

OPERATION REPORT
DRILL .203 DIA. .75 DEEP
TAP 1/4-20 .5 DEEP
COST 135



&ATTVL

Name **&ATTVL** Get Attribute Value

Synopsis **&ATTVL({obj|PART|'name'},'title'[,IFERR,label:][,data_type])**

Description Provides the only method of obtaining the value of an attribute. In addition, it is a unique and efficient method of editing the value of an existing attribute.

Characteristics Read/Write String 132 Characters

Parameters **obj**
Existing object whose attribute values will be accessed.

PART
Minor word which indicates that the part attributes will be accessed.

'name'
A string or string variable which represents the name of an object such as a reference set.

'title'
The attribute title whose corresponding value will be accessed.

[,data_type]
The parameter data_type has been assigned a value of GPA &ATTTYPE. If data_type is not specified, the default data type is variable length string. Note that you do not have to write anything to the Null attribute.

Part attributes can only have a data_type of 5 (string); the system ignores all other data_types for part attributes.

Data Type	Description
1	Integer
2	Floating point
3	Date and time
4	Null string
5	Variable length character
7	Reference



IFERR,label:

Specifies a label to which the program jumps if an error occurs. An error occurs when the specified title does not exist for the object.

Example

This example demonstrates the use of the **&ATTVL** EDA to extract values.

Declarations

```
GRIPSW/declrv
NUMBER/I
ENTITY/CR(3)
STRING/ATT(12,132),PR(6,132)
CR(1)=CIRCLE/1,1,.5
CR(2)=CIRCLE/2,1,.5
CR(3)=CIRCLE/3,1,.5
$$ DECLARE INTEGER ATTRIBUTE
ATT(1)='HOLE DEPTH'
ATT(2)='2'
$$ DECLARE FLOATING POINT ATTRIBUTE
ATT(3)='HOLE WIDTH'
ATT(4)='1.75'
$$ DECLARE DATE & TIME ATTRIBUTE
ATT(5)='DATE CREATED'
ATT(6)='06/25/92 14:35'
$$ DECLARE A NULL ATTRIBUTE
ATT(7)='NULL DATA'
$$ TO DECLARE A STRING ATTRIBUTE
ATT(9)='HOLE DESC'
ATT(10)='This hole was created '$
+ 'on jun 25, 92 at 2:35pm and has a NULL attribute.'
$$ DECLARE A REFERENCE
EXPCRE/'DEPTH = 2'
EXPCRE/'WIDTH = 1.75'
ATT(11)='HOLE DIMS'
ATT(12)='This hole is <X2.2@DEPTH> inches deep '$
+ 'and <X2.2@WIDTH> inches wide.'
ASATT/CR(1),ATT(1..2),1
ASATT/CR(2),ATT(3..4),2
ASATT/CR(3),ATT(5..6),3
ASATT/CR(3),ATT(7..8),4
ASATT/CR(3),ATT(9..10)
ASATT/CR(3),ATT(11..12),6
PR(1)=%ATTVL(CR(1),'HOLE DEPTH',1)
PR(2)=%ATTVL(CR(2),'HOLE WIDTH', 2)
PR(3)=%ATTVL(CR(3),'DATE CREATED',3)
```

```
PR(4)=&ATTVL(CR(3),'NULL DATA',4)
PR(5)=&ATTVL(CR(3),'HOLE DESC')
PR(6)=&ATTVL(CR(3),'HOLE DIMS')
PRINT/'HOLE DEPTH REPORT'
DO/L1:,I,1,6
PRINT/PR(I)
L1:
HALT
```

You can obtain the first title for CR(1) by using the attribute title EDA. That title can then be used to edit the value for the hole depth:

```
TLE=&ATTTL(CR(1),1,1)
&ATTVL(CR(1),TLE,1)='20'
```

DLATT

<i>Name</i>	DLATT	Delete Object Attribute
<i>Synopsis</i>	DLATT/{<u>obj_list</u> ALL PART 'name'},{<u>title list</u> ALL}[,<u>data_type</u>]	
<i>Description</i>	Deletes object attributes assigned to an object. Note that attributes on a tool object are not affected by this statement.	
<i>Parameters</i>	<p><u>obj_list</u> The list of objects from which the specified attributes are to be deleted. If the PART object identifier is included, it must be the first member.</p> <p>ALL Minor word which indicates that the specified attributes are to be deleted from all of the objects in the part (except manufacturing objects).</p> <p>PART Minor word which indicates that the attributes are to be deleted from the part instead of from individual objects.</p> <p>'name' A string or string variable which represents the name (object identifier) of an object such as a reference set.</p> <p><u>title list</u> The title list is a string list which consists of the attribute titles which are to be deleted from the specified objects.</p> <p>ALL Minor word which indicates that all of the attributes are to be deleted from the specified objects.</p> <p>[,<u>data_type</u>] The parameter data_type has been assigned a value of the GPA &ATTYPE. Depending on the value of data_type, object attributes for all specified objects are deleted. If data_type is not specified, the default data type is variable length string.</p> <p>Part attributes can only have a data_type of 5 (string); the system ignores all other data_types for part attributes.</p>	

Data_Type	Explanation
1	Integer
2	Floating point
3	Date and time
4	Null
5	Variable length character
6	All data types
7	Reference

Example This example demonstrates the use of the **DLATT** statement.

Declarations ENTITY/LN(4),CR(2)
STRING/ATT(6,132)

Geometry Definition LN(1) =LINE/-1,-1,1,-1
LN(2) =LINE/1,-1,1,1
LN(3) =LINE/1,1,-1,1
LN(4) =LINE/-1,1,-1,-1
CR(1) =CIRCLE/-.5,-.5,.25
CR(2) =CIRCLE/.5,.5,.25
ATT(1)='USE'
ATT(2)='PART EDGE'

Attribute Assignment ASATT/LN,ATT(1..2)

Reassign Value for ATT(2)='MOUNTING HOLE'
ATT(3)='DRILL'
ATT(4)='.5 DIA. .75 DEEP'
ATT(5)='TAP'
ATT(6)='9/16 18 THREAD'

Attribute Assignment ASATT/CR,ATT

Delete Attribute DLATT/LN(1..3),ALL \$All of the attributes are deleted from \$the lines in the object list LN(1..3).

DLATT/CR(1),ATT(1..2) \$ATT(1) and ATT(2) are deleted from \$CR(1).

DLATT/ALL,ALL,1 \$All integer attributes are deleted from \$all objects in the part.

DLATT/ALL,ALL \$All string attributes are deleted from \$all objects in the part.

DLATT/ALL,ALL,6

\$All attributes are deleted from all
\$objects in the part (except
\$manufacturing objects).



&ATDISL

Name **&ATDISL** Control Name Display Location (X,Y,Z)

Synopsis **&ATDISL(obj)**

Description Allows the origin coordinates of the object name, which are in the work coordinate system, to be obtained or altered. Display origins, which are determined automatically by the system at the time the name is established, are as shown below.

Object Type	Display Origin
Point	The point
Line	The midpoint
Arc	The midpoint
Conic	The midpoint
Spline	The first knot point
Surface	The midpoint
Group system	The first non-group member
Plane	The origin
Coordinate	The origin

Characteristics Read/Write Three Position Numerical Array (N(3))
 % Infinity

Parameters **obj**
 Existing object from which the name location will be accessed.

Example This example demonstrates the use of the **&ATDISL** EDA to extract the name display location of a line and alter that location.

Declarations ENTITY/LN1
 NUMBER/XYZ(3)
 LN1=LINE/-1,0,1,0
 XYZ=&ATDISL(LN1)

The variable array XYZ would be assigned the values (0,0,0) which is the midpoint of line LN1. The statements below alter the origin and cause the name to be displayed at the coordinates (.5,.5,0).



ASCII Conversion Table

Appendix A



ASCII CHARACTER	VALUE	ASCII CHARACTER	VALUE	ASCII CHARACTER	VALUE
(BLANK)	32	?	63	^	94
!	33	@	64	—	95
”	34	A	65	‘	96
#	35	B	66	a	97
\$	36	C	67	b	98
%	37	D	68	c	99
&	38	E	69	d	100
,	39	F	70	e	101
(40	G	71	f	102
)	41	H	72	g	103
*	42	I	73	h	104
+	43	J	74	i	105
,	44	K	75	j	106
—	45	L	76	k	107
.	46	M	77	l	108
/	47	N	78	m	109
0	48	O	79	n	110
1	49	P	80	o	111
2	50	Q	81	p	112
3	51	R	82	q	113
4	52	S	83	r	114
5	53	T	84	s	115
6	54	U	85	t	116
7	55	V	86	u	117
8	56	W	87	v	118
9	57	X	88	w	119
:	58	Y	89	x	120
;	59	Z	90	y	121
<	60	[91	z	122
=	61	\	92	{	123
>	62]	93		124
				}	125
				~	126



(This Page Intentionally Left Blank)

GRIP Debugger

Appendix B



GRIP Debugger

The **Debug GRIP** option allows you to debug GRIP code while running a program. The GRIP compiler and linker automatically generate debuggable code. Use the **Debug GRIP** execution option to run the GRIP debugger and identify further problems in the code.

You can use the GRIP debugger on any system that Unigraphics supports. The debugger is not available in GRIP batch because of its interactive nature, so it will not be available in any form on terminals that cannot run Unigraphics. However, you can batch compile and link code.

The GRIP debugger has capabilities which include manipulation of breakpoints, examination of variables, and control of program execution. All user interaction is done through menus.

Features of the Debugger

The debugger enables you to perform the following activities.

- Set, clear, and list break points at any line or label within the program.
- Examine the value of entity, number, string, or array variables.
- Change the value of a number or string variable.
- List all variables in a particular program or subprogram, and allow the user to look at the name, type, and dimensions of all variables in a subprogram or program.
- Control the program execution in the following ways:
 - single step into a called subroutine
 - single step over a called subroutine
 - continue from the breakpoint
 - abort the program.

The most important benefit to you, the programmer, is that by using breakpoints you can stop the program, and examine and modify number and string variables. You can examine, but not modify, entity variables.

Running a Program with the Debugger

When you choose **File**→**Execute/UG Open**→**Debug Grip**, the system prompts you to specify a file to debug. The **Compile** option on **GRADE** automatically generates debuggable code when compiling a GRIP program.

The GRIP Debugger Menu

Once you've specified a file to debug, the system displays the main **Debug GRIP** menu containing the following options:



Single Step
Single Step/Into
Set Breakpoints
Clear Breakpoints
Examine Variables/Entities
Set Variable
List Option
Continue
Program Abort

The menu header lists the following program information:

(subprogram) = name of the current subprogram
(line #) = the current line number of the program
(reason stopped) = the reason for suspending the program

The debugger will suspend execution for one of the following four reasons. Each is listed with its abbreviation as it appears on the menu:

Initial Entry (INIT ENTRY)
Breakpoint (BREAKPOINT)
Single Step (SINGLE ST.)
Single Step Into (Step/Into)

Single Step

The interpreter executes the current statement and suspends at the next line.

If the current instruction is a **CALL** statement, the interpreter executes all instructions and observes all breakpoints in the called subprogram. When control returns to the calling subprogram, the system sets a temporary breakpoint at the line following the **CALL** statement and suspends program execution. To resume program execution, choose either **Single Step** or **Continue** from the main *Debug GRIP* menu.

For example, if in the calling routine there is no breakpoint between the current line and the line that follows, the interpreter suspends the program due to **Single Step**. However, if there is a breakpoint(s) in between the two lines and the program execution suspends at the temporary breakpoint, the reason stopped is **Breakpoint**.

Single Step/Into

The interpreter executes the current statement and suspends at the next line.

If the current instruction is a **CALL** statement, the system suspends program execution at the first executable statement in the called subprogram. The reason stopped is **Step/Into**.

Single Step and **Single Step/Into** perform exactly the same for all non-**CALL** statements.

Set Breakpoints

Allows you to set breakpoints at either specific line numbers or labels within a specified program. By setting breakpoints, you can suspend program execution at specified locations. You can then examine the status of the program by using other debugger options.

The maximum number of breakpoints you can set in one program is 26. When you choose this option, the system prompts you with the following options:

PROG_NAME
Line/Label

PROG_NAME allows you to enter the name of a subprogram in which you wish to set a breakpoint. The maximum length of the name including the extension is 30 characters. The default is the current subprogram name.

If the system cannot find the input name, it displays the message

Undefined Subprogram Name

Line/Label allows you to specify the line number or label name where you wish to set the breakpoint.

If you enter a line number:

- and the input line number is less than the line number of the first executable statement in the program, the system displays the line number of the first executable line and sets a breakpoint at that line.
- and the input line number is greater than the last line in the program, the system displays the line number of the last executable line.
- and the input line number is at a comment, the system displays the line number of the two closest executable lines, one before and one after the comment.

If you enter a label name, the system sets a breakpoint at that label. You can enter the label name either by itself (**END**) or with a colon (**END:**). The length of an input label can be more than 6 characters, but the system only uses the first 6 characters, so they should be unique.

If the system cannot find the label name, it displays the error message

Undefined Label Name

NOTE There is no line number assigned to DO loop labels in the object code. Therefore, in order to set a breakpoint at a DO loop label, you must have a dummy statement at the DO loop termination.



Clear Breakpoints

Allows you to remove any or all breakpoints previously set in the program. If you have not previously set any breakpoints, the system displays the error message

There Is No Breakpoint To Clear

and returns to the main *Debug GRIP* menu.

If there are one or more breakpoints set in the program, the system displays a list of the current breakpoints set in the program.

Choosing **All** clears all breakpoints, including any temporary breakpoints, from the program. The system then returns to the main *Debug GRIP* menu.

If there are 13 or more breakpoints in the program, **Next Page** will appear as the final option allowing you to continue on through the remaining breakpoints.

Choosing one or more breakpoints clears those selected breakpoints from the program.

Examine Variables/Entities

Allows you to display the values of variables and/or entities in a specified program. When you choose this option, the system displays the following options:

PROG_NAME
VAR_NAME

PROG_NAME allows you to enter the name of the subprogram in which you wish to examine variables. If no name is entered, the system defaults to the current subprogram.

VAR_NAME allows you to enter the name of the variables/entities you wish to examine. Based on the type of variable/entity specified, the system sends the following information to the listing window.

- Variable/Entity name by itself:
 - Numeric variable: subprogram name, variable name, real value
 - String variable: subprogram name, variable name, string value
 - Numeric array variable: subprogram name, variable name, indices, real value at each index
 - String array variable: subprogram name, variable name, indices, string value at each index
 - Single entity: subprogram name, entity name, value (either null entity, the value of the valid entity, or invalid entity)
 - Entity array: subprogram name, entity name, indices, value at each index (either null entity, the value of the valid entity, or invalid entity)
- Variable/Entity name with an index: If a variable/entity is not an array type, or the input index is out of range, the system displays an error message. When examining an index of a variable/entity there should only be one open and one closed parenthesis, and 0, 1, or 2 commas, depending on the variable/entity dimension. The index values must be constants.
 - Numeric array variable: subprogram name, variable name, index and real value of that index
 - String array variable: subprogram name, variable name, index and string value of that index
 - Entity array: subprogram name, entity name, index, value of that index (either null entity, the value of the valid entity, or invalid entity)
- Variable/Entity name with a subrange: If a variable/entity is not an array type, or the input subrange is out of range, the system displays an error message. When examining a subrange of a variable/entity, there should only be one open and one closed parenthesis, and double periods (..) between the lower and upper bounds, as well as 0, 2, or 4 commas depending on the variable/entity dimension. The lower and upper bound values must be constants.
 - Numeric array variable: subprogram name, variable name, indices from lower to upper range, real value of each index



- String array variable: subprogram name, variable name, indices from lower to upper range, string value of each index
- Entity array: subprogram name, entity name, indices from lower to upper range, value of each index (either null entity, the value of the valid entity, or invalid entity)

Set Variable

Allows you to set a variable one value at a time; you cannot set a range using this option. When you choose this option, the system displays the following options:

PROG_NAME
VAR_NAME

PROG_NAME allows you to enter the name of the subprogram in which you wish to set the variable. If no name is entered, the system defaults to the current subprogram.

VAR_NAME allows you to enter the name of the variable you wish to set.

If the specified variable is a number, the system sends the subprogram name, variable name, and its original value to the listing device. If the variable is a numerical or string array, the system sends the variable name, index, and original value of that index. The system then prompts you to enter a new value.

If the specified variable is a string, the system sends the subprogram name, variable name, its original value, and (for arrays) any indices and their values to the listing device. The system then prompts you to enter a new string.

After specifying the new variable, the system sends the new value to the listing device and returns to the **Set Variable** menu.

List Options

Allows you to list subprograms, and breakpoints contained in the program. When you choose this option, the system displays the following options:

List Variables in Subprogram

List All Variables

List All Subprograms

List Breakpoints

List Variables

List Variables in Subprogram allows you to list all the variable names, types, and dimensions of a specified subprogram. When you choose this option, the system prompts you to enter the name of the subprogram.

The system defaults to the name of the current subprogram. All variable names, types and dimensions in the specified subprogram are sent to the current listing device. If the variable type is a number or label, the system only sends the name and the type.

List All Variables

List All Variables allows you to list all variable/label names, types, and dimensions of ALL subprograms. The system sends the information to the current listing device, starting with the main program and continuing to the last subprogram. The system lists each program name, followed by the variable names, types, and dimensions in each subprogram. If the variable type is a number or label, then only the name and type is listed.

List All Subprograms

List All Subprograms allows you to send to the current listing device all subprogram names, along with the name of the main program.

List Breakpoints

List Breakpoints sends a list of all breakpoints to the current listing device. The system sends the module name and the line number of each breakpoint.

Continue

This option continues program execution until the next breakpoint or the end of the program.

Program Abort

This option aborts program execution.





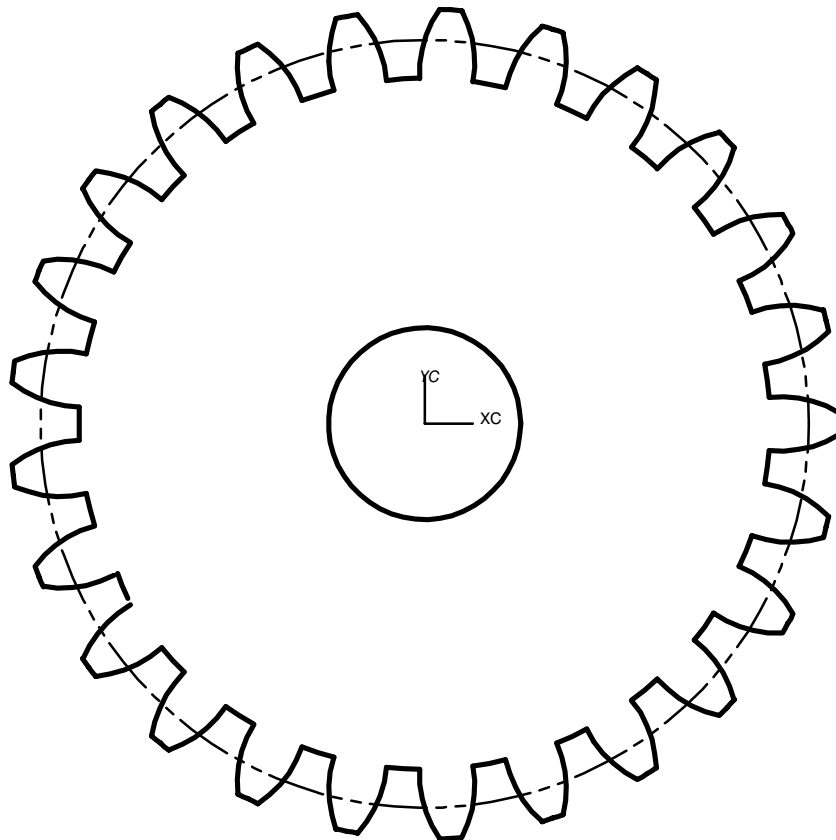
(This Page Intentionally Left Blank)

Final Project
Appendix C



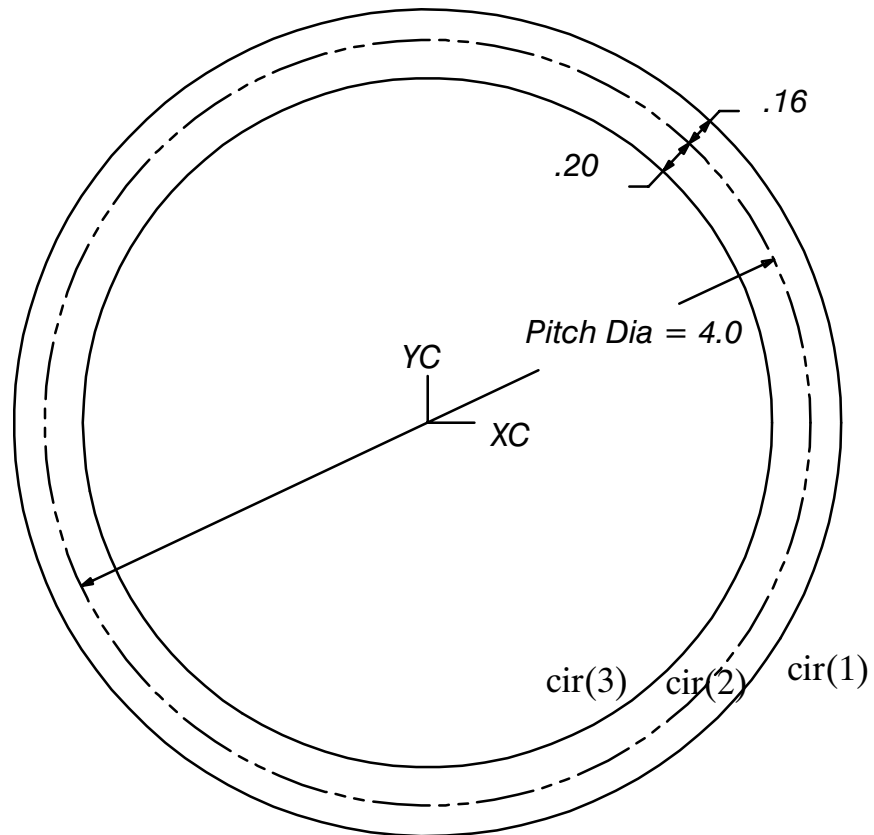
Activity: Creating a Spur Gear

Use curve creation and transformation commands to construct the profile of the spur gear shown below.

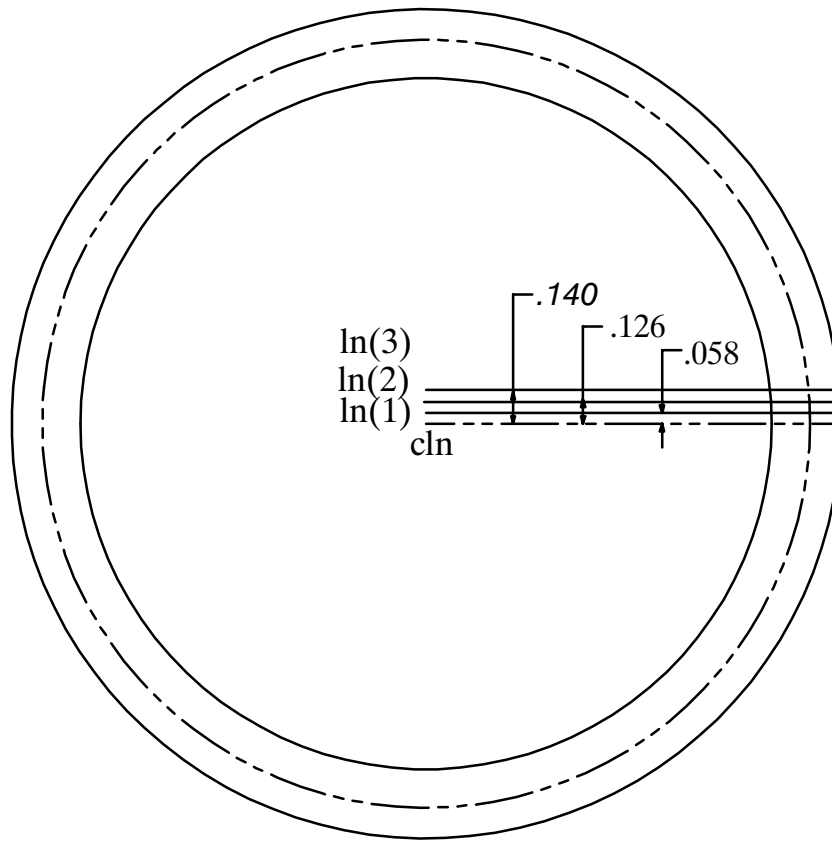


TIP A “Best Practice” is to compile, link and run the program often. This will highlight the functionality and effect of small sections of code. This incremental approach to coding also allows easier debugging of a program.

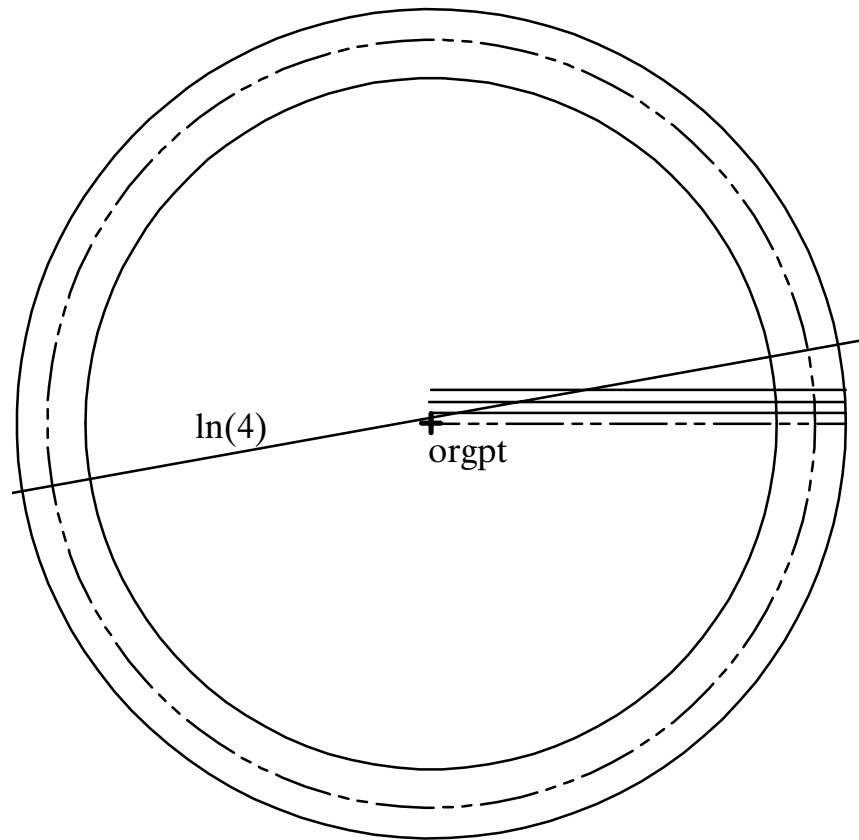
Step 1 Create the arcs to define half of a gear tooth. There will be a total of 6 arcs per tooth. Only three arcs are created explicitly; the other three are mirrored.



- Create the addendum (outer), circle, pitch circle, and the dedendum (inner) circle. The outer and inner circles will be trimmed to form two of the arcs needed for the tooth. The pitch circle will be used for construction purposes and deleted.

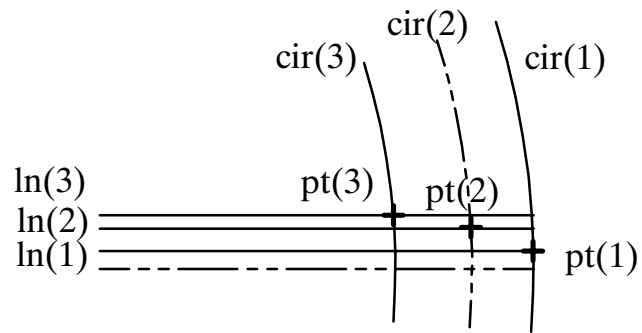


- Create a horizontal line along the XC-Axis representing the centerline of the first tooth. Create three additional lines parallel to the centerline with the distances shown.

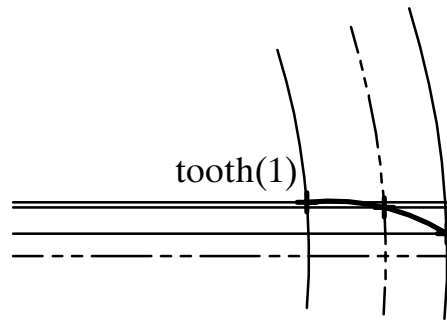


- Create a point at the WCS origin (orgpt). Create a line (ln(4)) through the origin point at half the angle of rotation for 25 teeth ($360/(2*25)$).



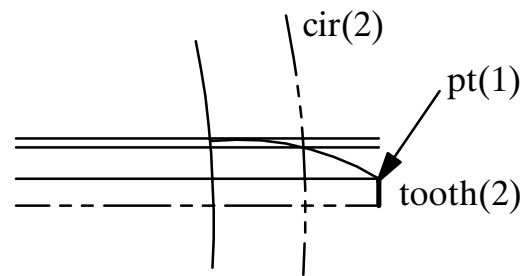


- Create three points at the intersections of the horizontal lines and the circles. These points will be used to create the side of the tooth.

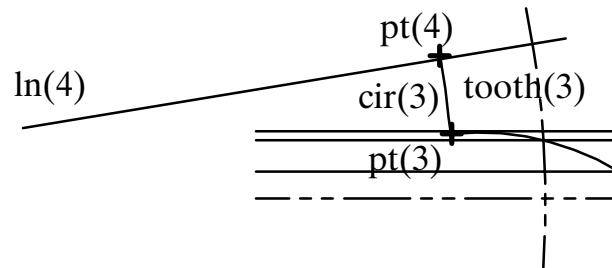


- Create the first arc, tooth(1), through three points for the side of the tooth.



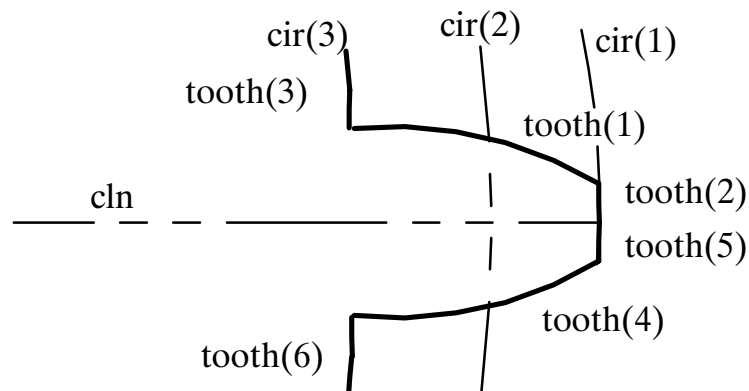


- Trim the outer circle (cir(1)) to end at pt(1). The start point will remain unchanged. Use the &EPOINT EDA to edit the end point of this arc to the existing point pt(1). This arc will be the end of the tooth.



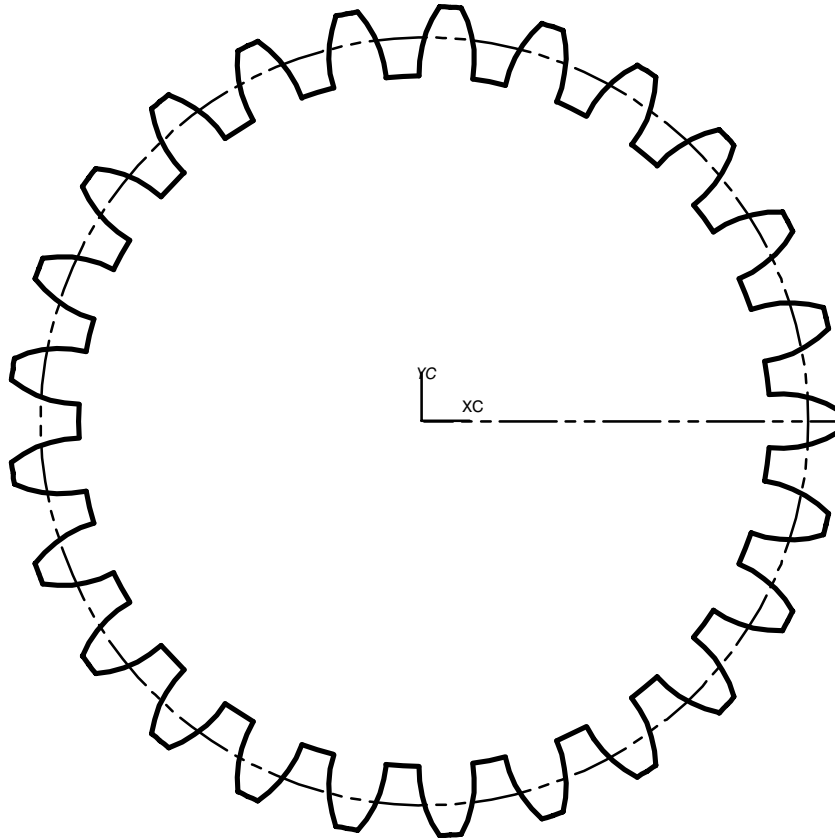
- Delete two of the three intersection points (pt(1) and pt(2)) as they are no longer needed. Create a new intersection point using the line at an angle, ln(4), and the inner circle, cir(3). Trim the inner circle by changing both the start and end points. Use the &SPOINT and &EPOINT EDAs to convert the circle to an arc.

Step 2 Apply a mirror transformation to the three arcs about the horizontal centerline to create the complete tooth.



- ❑ Create the mirror matrix, then use TRANSF/ to mirror the tooth arcs.

Step 3 Apply a rotation transformation to the six arcs representing the tooth to create the other 24 teeth.

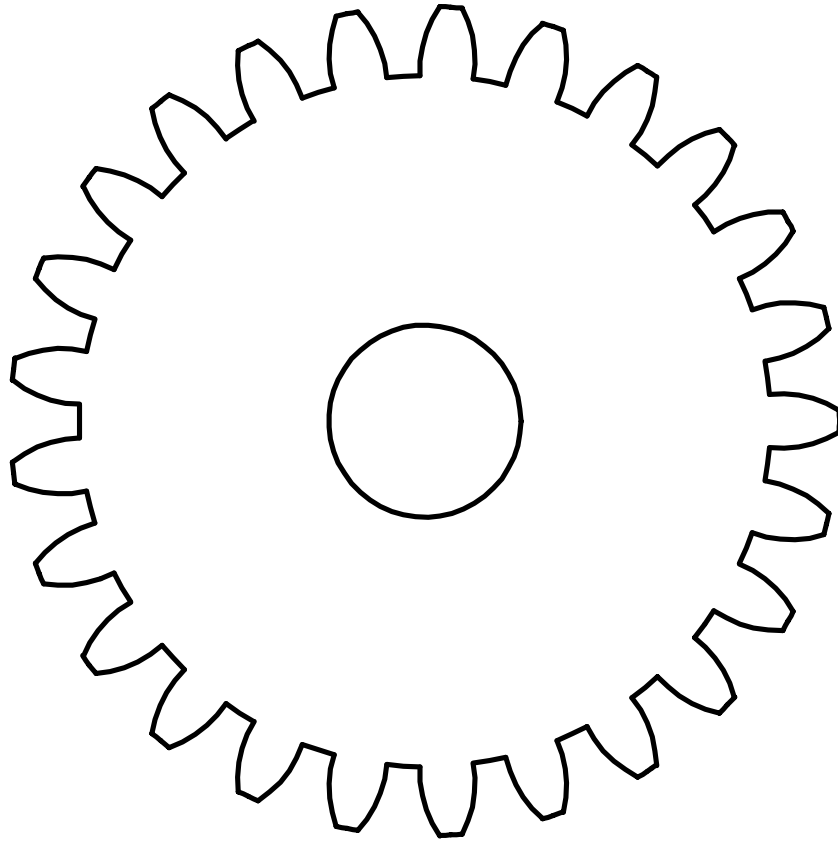


- ❑ Create a rotation matrix, then use TRANSF/ to rotate the tooth arcs. You can place the transformation command in a DO loop to create multiple copies.

TIP Use a two-dimensional array for the arc objects used in the tooth. The array should be 25x6 to hold the six arcs needed for the 25 teeth



Step 4 Create a 0.5 inch radius circle for the hole and delete any unwanted construction geometry.



Optional Steps

- Create an extruded solid body using the gear profile curves. Use a thickness of 1 inch.

NOTE When you extrude a solid, edits to the base curves such as the hole circle will change the solid. If you create the hole using a boolean operation to subtract a cylinder, you will not be able to change the hole diameter as easily.

- Add .03 inch fillet curves at the base and top of the teeth. Remember to include the fillets in the transformation and extruded body commands.

(This Page Intentionally Left Blank)



Statement Format Summary

Appendix D

This appendix contains an alphabetical list of all GRIP statements. This listing is purely alphabetical, by major word, and assumes that you know the spelling of the major word you are looking for. You may find this appendix most helpful once you are familiar with the GRIP language and only need to reference the format of certain statements from time to time.

Similar format summaries for GPAs and EDAs can be found in later sections.

Function	Statement Format
ABSOLUTE VALUE OF arg	ABSF(arg)
ANGLE WHOSE COSINE IS arg	ACOSF(arg)
ADD ROW TO PARTS LIST	ADDPL/{num list,string list[{,INT ,STR}], quantity <u>obj list</u> } [,IFERR,label:]
ANGULAR DIMENSION	<u>obj</u> = ADIM/[MAJOR,](origin),”PMOD3”,line1, [VIEW,’View Name’,]”PMOD3”,line2 [,VIEW,’View Name’] [,Dim. text][,APPEND,App. text]
ANGLE OF A LINE	ANGLF({ <u>line</u> <u>circle1</u> , <u>circle2</u> <u>point1</u> , <u>point2</u> })
2D ANALYSIS	ANLSIS/TWOD[,TOLER,t], <u>obj list</u> , {INCHES MMETER CMETER METER} ,n(14)
3D ANALYSIS (Rotate About XC or YC Axis)	ANLSIS/VOLREV,{XAXIS YAXIS},d[,TOLER,t] , <u>obj list</u> ,{INCHES MMETER CMETER METER},n(41)
3D ANALYSIS (Project Along ZC Axis)	ANLSIS/PROSOL,d,lim1,lim2[,TOLER,t] , <u>obj list</u> ,{INCHES MMETER CMETER METER},n(41)
3D ANALYSIS (Bounded by Faces)	ANLSIS/VOLBND,d[,ACCRCY,a ,TOLER,t], <u>obj list</u> , {LBIN LBFT GCM KGM},n(41)



Statement Format Summary

Function	Statement Format
3D ANALYSIS (Thin Shell)	ANLSIS/SHELL,d[,ACCRCY,a ,TOLER,t], <u>obj list</u> , {LBIN LBFT GCM KGM},n(41)
ARC LENGTH	ANLSIS/ARCLLEN[TOLER,t], <u>obj list</u> , {INCHES MMETER CMETER METER},n
MASS PROPERTIES OF SOLID BODIES	ANLSIS/SOLID[,ACCRCY,a ,TOLER,t], <u>entlist</u> {,LBIN ,LBFT ,GCM ,KGM},n
FILE POINTER CONTROL	APPEND/file#
ARC LENGTH DIMENSION	<u>obj</u> = ARCDIM/(origin),Arc[,VIEW,'View Name'] [,Dim. text][,APPEND,App. text]
ASSOCIATED ENTITY OF DIM/DRAFT ENTITY	ASCENT/ <u>ent</u> ,n, <u>assoc. ent</u> [,assoc. type] [,assoc. modifier] [,IFERR,label:]
RETURN ASCII VALUE OF CHARACTER	ASCII('string',pos)
ASSIGN ATTRIBUTES	ASATT/{ <u>obj list</u> ALL PART 'name'},attribute list[,data_type]
ASSIGN FONT	ASGNFT/name,number[,IFERR,label:]
ANGLE WHOSE SINE IS arg	ASINF(arg)
ANGLE WHOSE TANGENT IS arg	ATANF(arg)
AUTOMATIC SURFACING	<u>obj</u> = AUTOSF/ <u>entlist</u> [,BYLAYR][,CNT,count] [,IFERR,label:]
IDENTIFY BASE FACE	<u>obj</u> = BASURF/ <u>ent</u> [,IFERR,label:]
COMPILE, LINK, OR RUN	num list = BATCH/{COMPIL LINK RUN},file list [,LP OS NULL] [, 'filespec'] [,QUEUE,queuename] [,STR,stringdata] [,IFERR,label:]



Function	Statement Format
CANCEL A BATCH JOB	num list = BATCH/CANCEL,job number list [,IFERR,label:]
B-CURVE (Fit Method)	<u>obj list</u> = BCURVE/FIT,{ <u>obj list</u> ,num list1} [,WGHT,num list2], {SEGS TOLER},num1[,DEGREE,num2] [,START,{VECT,dx,dy,dz TANTO, {curve angle}}] [,END,{VECT,dx,dy,dz TANTO, {curve angle}}] ,STATUS,numa[,IFERR,label:]
B-SPLINE (Point Method)	<u>obj list</u> = BCURVE/ <u>entlist</u> [,VERT[,numlist]] [,DEGREE,num[,CLOSED]] [,IFERR,label:]
B-SPLINE (Curve Method) +	<u>obj list</u> = BCURVE/ <u>entlist</u> ,ENDOF {,entlist2 ,numlist} [,DELETE ,BLANK] [,IFERR,label:]
BLANK	BLANK/{ <u>obj list</u> ALL}
BLEND/CHAMFER SOLID EDGES	BLEND/ <u>ent</u> ,{RADIUS CHAMFR},num [, <u>entlist1</u>] [,VERT, <u>entlist2</u>] [,IFERR,label:]
FIX BLEND/CHAMFER	BLENFX/ <u>entlist</u> [,IFERR,label:]
CREATE BLANK CHARACTERS	BLSTR(n)
CREATE BOUNDARY ENTITY	<u>obj list</u> = BOUND/[CLOSED OPEN,] [TOLER,intol,outtol,] [,{ON TANTO,} <u>entlist</u>] + [VIEW, 'View Name']
BOUNDED PLANE	<u>obj</u> = BPLANE/ <u>obj list1</u> [,HOLE,nlist, <u>obj list2</u>] [,TOLER,t]
B-SURFACE (Through Points)	<u>obj</u> = BSURF/ <u>obj list</u> ,num list1[,VERT[,num list2]] [,DEGREE,num1[,CLOSED],num2 [,CLOSED]][,IFERR,label:]



Function	Statement Format
B–SURFACE (Through Curves)	<u>obj</u> = BSURF/CURVE, <u>obj list1</u> [,ENDOF{, <u>entlist2</u> ,numlist}] [,DEGREE,num[,CLOSED]] [,IFERR,label:]
B–SURFACE (Conic)	<u>obj</u> = BSURF/CONSRF,num1, <u>entlist</u> ,SPINE, <u>ent1</u> [,ENDOF, <u>ent2</u>][,RHO,nlist] [,TOLER,num2][,APEX, <u>ent3</u>] [,RESULT,num3] [,IFERR,label:]
B–SURFACE (Through Curve Mesh)	<u>obj</u> = BSURF/MESH, <u>entlist1</u> ,WITH, <u>entlist2</u> [,TYPE,num1] [,TOLER,num2,num3] [,RESULT,num4] [,IFERR,label:]
B–SURFACE (Swept)	<u>obj</u> = BSURF/SWPSRF ,TRACRV, <u>entlist1</u> [,ENDOF, <u>entlist2</u>] ,GENCRV, <u>entlist3</u> [,ENDOF, <u>entlist4</u>] [,BLEND,num1] [,SPINE, <u>ent1</u> [,ENDOF, <u>ent2</u>]] [,ORIENT{, <u>ent3</u> [,ENDOF, <u>ent4</u>] ,xc,yc,zc}] [,SCALE{, <u>ent5</u> [,ENDOF, <u>ent6</u>] ,nlist}] [,TOLER,num2,num3][,RESULT,num4] [,IFERR,label:]
B–SURFACE (Convert Existing Surface)	<u>obj</u> = BSURF/SURFC, <u>ent</u> [,APPROX] [,TOLER,dtol,atol] [,IFERR,label:]
CALL SUB–ROUTINE	CALL/'subprogram name'[,actual argument list]
CREATE CATEGORY	CAT/'name'[,layer list][,CAT,'cat'] [,DESCR,'description']
EDIT CATEGORY	CATE/'name'[,ADD ,REMOVE] [,layer list][,CAT,'cat list'] [,DESCR,'description'][,IFERR,label:]
DELETE CATEGORY	CATD/'name'[,IFERR,label:]
QUERY CATEGORY	CATV/'name'[,LAYER,layers,CNT,count] [,DESCR,'description'][,IFERR,label:]



Function	Statement Format
CONCENTRIC CIRCLES DIMENSION	<u>obj</u> = CCDIM/(origin),arc1,[VIEW,'View Name',]arc2 [,VIEW,'View Name'],{LEFT RIGHT},] [,Dim. text],[APPEND,App. text]
CHAIN SELECT	CHAIN/START, <u>ent1</u> [,{"PMOD3" <u>point</u> }] [,END, <u>ent2</u>], <u>ent array</u> [,CNT,count] [,IFERR,label:]
CHECK VALIDITY OF A SOLID	CHKSOL/ <u>entlist</u> ,RESULT,nlist [,IFERR,label:]
CURVE HIDING CURVE	CHIDC/ <u>obj list</u> [,IFERR, label:]
CHOOSE OPTIONS	CHOOSE/string list,[DEFLT,n,] [ALTACT,'message',] response
RETURN STRING WITH ASCII VALUE OF n	CHRSTR(n)
CIRCLE (Center Point, Radius)	CIRCLE/CENTER, <u>point</u> ,RADIUS,r [,START,start angle,END,end angle]
(Center Point, Tangent To A Line)	CIRCLE/CENTER, <u>point</u> , TANTO, <u>line</u> [,START,start angle,END,end angle]
(Center Point, Point On Arc)	CIRCLE/CENTER, <u>point1</u> , <u>point2</u> [,START,start angle,END,end angle]
(Through Three Points)	CIRCLE/ <u>point1</u> , <u>point2</u> , <u>point3</u>
(Center Coords, Radius)	CIRCLE/x,y,[z,]r [,START,start angle,END,end angle]
FULL BOLT CIRCLE CENTERLINE	<u>obj</u> = CLINE/FBOLT[,CENTER, <u>obj</u> [,VIEW,'View Name']], <u>obj list</u> [,VIEW,{'View Name' view name list}]
FULL CIRCULAR CENTERLINE	<u>obj</u> = CLINE/FCIRC [,CENTER, <u>obj</u> [,VIEW,'View Name']] <u>obj list</u> [,VIEW,{'View Name' view name list}]
LINEAR CENTERLINE	<u>obj</u> = CLINE/LINEAR, <u>obj list</u> [,VIEW,view name list]



Function	Statement Format
OFFSET CENTER POINT (Format 1)	<u>obj</u> = CLINE/OFFCPT,{XCAXIS YCAXIS} ,CENTER, <u>obj1</u> ,[VIEW,'View Name'],arc [,VIEW,'View Name']
(Format 2)	<u>obj</u> = CLINE/OFFCPT,{XCAXIS YCAXIS},DSTCTR ,num,arc[,VIEW,'View Name']
(Format 3)	<u>obj</u> = CLINE/OFFCPT,{XCAXIS YCAXIS} ,DSTNRM,num,arc[,VIEW,'View Name']
OFFSET CYLINDRICAL CENTERLINE (Format 1)	<u>obj</u> = CLINE/OFFCYL,OFFDST,num, <u>obj1</u> , [VIEW,'View Name'], <u>obj2</u> [,VIEW,'View Name']
(Format 2)	<u>obj</u> = CLINE/OFFCYL,OFFPT, <u>obj1</u> , [VIEW,'View Name'], <u>obj2</u> , [VIEW,'View Name'], <u>obj3</u> [,VIEW,'View Name']
PARTIAL BOLT CIRCLE CENTERLINE	<u>obj</u> = CLINE/PBOLT[,CENTER, <u>obj</u> [,VIEW,'View Name']] ,obj list[,VIEW,view name list]
PARTIAL CIRCULAR CENTERLINE	<u>obj</u> = CLINE/PCIRC [,CENTER, <u>obj</u> [,VIEW,'View Name']] ,obj list[,VIEW,view name list]
SYMMETRICAL CENTERLINE	<u>obj</u> = CLINE/SYMMET, <u>obj1</u> [,VIEW,'View Name'], <u>obj2</u> [,VIEW,'View Name']
COMPARE STRINGS	CMPSTR('string1','string2')
CYCLE OBJECTS IN A COMPONENT	<u>obj</u> = CNEXT/component_obj_id,current_object [,IFERR,label:]
CONE (Circle, Height, Vertex Half-angle)	<u>obj</u> = CONE/arc,{"PMOD3" <u>point</u> },d,ANGLE,a
(Two Circles)	<u>obj</u> = CONE/ <u>arc1</u> , <u>arc2</u>
(Center Point, Existing Line)	<u>obj</u> = CONE/CENTER, <u>point</u> ,[VECT,x,y,z,] <u>line</u>



Function	Statement Format
(Vertex Point, Vertex Half-angle, Bounding Points)	<code>obj = CONE/point1,[VECT,x,y,z,] ANGLE,a,point2,point3</code>
COSINE OF angle	<code>COSF(angle)</code>
PARAMETER POSITION ON A CURVE	<code>num = CPARF/ent,{point x,y,z}</code>
EXPORT A FILE	<code>CPATT/[UPDATE,]'filespec'[,CSYS,csys] [,ORIGIN,point],obj_list[,IFERR,label:]</code>
POSITION ON A CURVE OR CURVE EXTENSION	<code>CPOSF(ent,scalar)</code>
GEOMETRIC PROPERTIES OF A CURVE AT PARAMETER	<code>CPROPF(obj,parameter)</code>
POINT SETS (Chordal Tolerance Method)	<code>CPSET/CHORD,obj,tolerance,results</code>
POINT SETS (Equal Parameter Method)	<code>CPSET/EPARAM,ent,n [,PART,a,b],results</code>
(Equal Arc Method)	<code>CPSET/EARCL,ent,n[,PART,a,b],results</code>
(Input ArcLength Meth)	<code>CPSET/ARCLen,ent,arclength,results</code>
(Geometric Progression Method)	<code>CPSET/GEOM,ent,n,RATIO,r[,PART,a,b],results</code>
(Control Vertex Method)	<code>CPSET/VERT,ent,results</code>
(Knot Point Method)	<code>CPSET/KNOT,ent,results</code>
CREATE DIRECTORY	<code>CRDIR/'filespec'[,IFERR,label:]</code>
CREATE A FILE	<code>CREATE/{PART,'filespec' {,INCHES MMETER} TXT,file# [,number list][,'filespec']} [,IFERR,label:]</code>



Statement Format Summary

Function	Statement Format
CROSS PRODUCT OF TWO VECTORS	CROSSF(vector1,vector2)
CREATE A REFERENCE SET	CRRFST/'reference set name', <u>obj list</u> [,CSYS,csys][,ORIGIN, <u>point</u>]
CREATE SOLID EDGES AND SILHOUETTES IN WORK VIEW	CRSEWV/[TOLER,t][,IFERR,LABEL:]
TRIM CURVE USING BOUNDING ENTITIES	<u>obj</u> = CRVTRM/curve,REF,{pt1 x,y,z},FIRST,limit1 [,REF,{pt1 x,y,z}] [,INT,{pt2 x,y,z}] [,NOTRIM] [,SECOND,limit2[,REF,{pt1 x,y,z}] [,INT,{pt2 x,y,z}][,NOTRIM]] ,STATUS,status [,IFERR, label:]
WRITE TO SCREEN	CRTWRT/'message',x,y,z
CSYS BY ORIGIN, X-PT, Y-PT	<u>obj</u> = CSYS/ <u>point1</u> , <u>point2</u> , <u>point3</u> [,ORIGIN, <u>point</u>]
CSYS BY X-AXIS, Y-AXIS	<u>obj</u> = CSYS/ <u>line1</u> , <u>line2</u> [,ORIGIN, <u>point</u>]
CSYS BY X-PT, Z AXIS	<u>obj</u> = CSYS/ <u>point</u> , <u>line</u> [,ORIGIN, <u>point</u>]
COORDINATE SYSTEM OF AN ARC	<u>obj</u> = CSYS/ <u>arc</u> [,ORIGIN, <u>point</u>]
COORDINATE SYSTEM OF A CONIC	<u>obj</u> = CSYS/ <u>conic</u> [,ORIGIN, <u>point</u>]
EXISTING COORDINATE SYSTEM	<u>obj</u> = CSYS/ <u>coordinate system</u> [,ORIGIN, <u>point</u>]
COORDINATE SYSTEM OF VIEW	<u>obj</u> = CSYS/{view number 'view name'} [,ORIGIN, <u>point</u>]
VECTOR TANGENT TO A CURVE	CTANF(<u>ent</u> ,scalar)



Function	Statement Format
TRIM CURVE BY GIVEN ARCLength	CTRIM/ <u>ent</u> ,dist,{START END <u>point</u> } or CTRIM/TOTAL, <u>ent</u> ,length,{START END <u>point</u> }
CYLINDRICAL DIMENSION	<u>obj</u> = CYLDIM/(origin), [{ENDOF CENTER TANTO},] "PMod3",obj1,[VIEW,'View Name',] [{ENDOF CENTER TANTO},]"PMod3", obj2,[VIEW,'View Name'][,Dim. text] [,APPEND,App. text]
CYLINDER (Arc, Bounding Plane)	<u>obj</u> = CYLNDR/ <u>arc</u> , <u>plane</u> , <u>point</u>
(Center Point, Line)	<u>obj</u> = CYLNDR/CENTER, <u>point</u> , <u>line</u>
(Center Point, Radius)	<u>obj</u> = CYLNDR/ <u>point</u> ,RADIUS,r
(Center Point, Radius, Two Bounding Planes)	<u>obj</u> = CYLNDR/ <u>point1</u> ,[VECT,x,y,z,]RADIUS,r , <u>plane1</u> , <u>plane2</u> , <u>point2</u>
(Radius, Between Two Surfaces)	<u>obj</u> = CYLNDR/ <u>surf1</u> , <u>surf2</u> ,CENTER, <u>point1</u> , RADIUS,r, <u>plane1</u> , <u>plane2</u> , <u>point2</u>
INITIALIZE VALUES FOR STRING AND NUMERICAL VARIABLES	DATA/name,value[,value] + [,name,value[,value] +] +
RETURN CURRENT DATE	DATE
CLOSE DIRECTORY	DCLOSE[/IFERR,label:]
DIAMETER	<u>obj</u> = DDIM/(origin),arc[,VIEW,'View Name'] [,Dim. text][,APPEND,App. text]
DELETE ENTITIES	DELETE/{ <u>obj list</u> ALL}
CHANGE DELIMITER	DELIM/'character'
DELETE OBJECT NAME	DELNAM/{ <u>obj list</u> ALL}



Statement Format Summary

Function	Statement Format
SET ENTITY LINE WIDTH	DENS/{NORM HEAVY THICK THIN}
SET DEPTH	DEPTH/z
DEVIATION CHECKING	DEVCHK/ent1[,ent1a],TO,ent2[,p1[,p2]] [,TOLER,t1[,t2]]
SPECIAL CONTROL FUNCTIONS	DFSTR/(num)
ADD SYMBOL TO DRAFTING ENTITY	DFTSYM/'symbol_name',ent,point, {SCALE,scale[,RATIO,ratio] SIZE,length,height} [,IFERR,label:]
CREATE A DIAGRAM	DIAGM/'os filespec','drawing name' [,{,COLOR,pens ,LWIDTH,pens , DENS,pens}] [,{,SCALE,scale ,SIZE,x,y}] [,IFERR,label:]
DIMENSION BY PARTS	obj = DIMBP/{obj list comp list obj list,comp list}
SET DIMENSION PARAMETERS BY DRAFTING ENTITY	DIMPAR/[DRAW,]ent
MINIMUM DISTANCE	DISTF({point line},{line point})
DELETE ATTRIBUTE	DLATT/{obj list ALL PART 'name'}, {title list ALL}[,data_type]
READ HEADER NEXT OF FILE IN DIRECTORY	DNEXT/IFEND,label:[,IFERR,label:]
PROGRAM LOOP	DO/label:,index variable,start,end[,increment]
OPEN DIRECTORY	DOPEN['filespec'][,IFERR,label:]
DOT PRODUCT OF TWO VECTORS	DOTF(A,B)
DRAW ON/OFF	DRAW/{ON OFF ALL obj list}



Function	Statement Format
CREATE A DRAWING	DRAWC/'drawing name', [MMETER,]{height,width n} [,IFERR,label:]
DELETE A DRAWING	DRAWD/'drawing name'[,IFERR,label:]
EDIT A DRAWING (Add View)	DRAWE/['drawing name',]ADD,'view name',x,y [,IFERR,label:]
(Relocate View)	DRAWE/['drawing name',] MOVE,'view name',x,y [,IFERR,label:]
(Remove View)	DRAWE/['drawing name',]REMOVE,'view name' [,IFERR,label:]
(Change DWG Size)	DRAWE/['drawing name',]SIZE, [{INCHES MMETER},] {height,width n} [,IFERR,label:]
(Change View Status)	DRAWE/['drawing name',]DVSTAT,'viewname', {REF ACTIVE} [,IFERR,label:]
RENAME A DRAWING	DRAWN/['old drawing name',] 'new drawing name' [,IFERR,label:]
VERIFY DRAWING	DRAWV/['drawing name',] [,{PLOT DVSTAT,numlist,},] variable list [,IFERR,label:]
DEFINE VIEW	num = DVIEW/ <u>coordinate system</u>
CREATE EDGE VERTEX POINTS	<u>obj list</u> = EDGVER/ <u>ent</u> [,CNT,c][,IFERR,label:]
EDIT ROW OF PARTS LIST	EDITPL/{kv1,kv2,kv3 <u>ent id</u> } [,num list,string list] [,QTY[,{,INT ,STR}],quantity] [,IFERR,label:]



Statement Format Summary

Function	Statement Format
ADD TO OR REMOVE FROM A REFERENCE SET	EDRFST/'reference set name', {APPEND DELETE}, <u>obj list</u>
EDIT CROSSHATCHING PARAMETERS	EDTXHT/ent, {BND, {ADD, <u>entlist</u> REMOVE, <u>entlist</u> REPLAC, <u>ent, boundary</u> } XHATCH [, FNAME, 'filename'] [, UTIL, {material number 'material name'}, angle, distance AFILL, fill number, angle, scale} [, IFERR, label:]
PRINT AT THE TOP OF THE NEXT PAGE	num = EJECT/{PRINT WINDOW}
ELLIPSE	<u>obj</u> = ELLIPS/ <u>point</u> , semimajor, semiminor [, ATANGL, angle] [, START, angle, END, angle]
DECLARE ENTITY VARIABLE	ENTITY/name[(dim1[, dim2[, dim3]])] [, name[(dim1[, dim2[, dim3]])]] +
SOLID CONTAINMENT	num = ENCONT/ <u>ent1</u> , <u>ent2</u> , [, IFERR, label:]
ACCESS ENVIRONMENT VARIABLES	string = ENVVAR/'variable' {, ASK , SET, 'value'} [, IFERR, label:]
VERIFY ATTRIBUTE NAME	num = ENUM/'name'
EXPONENTIAL FUNCTION arg	EXPF(arg)
CREATE EXPRESSION	EXPCRE/exp_string[, IFERR, LABEL:]
DELETE EXPRESSION	EXPDEL/name_string[, IFERR, LABEL:]
EDIT EXPRESSION	EXPEDT/exp_string_list[, IFERR, LABEL:]
EVALUATE EXPRESSION	num = EXPEVL/name_string[, IFERR, LABEL:]
EXPORT EXPRESSION	EXPEXP/file_string[, IFERR, LABEL:]
IMPORT EXPRESSION	EXPIMP/file_string[, REPLAC][, IFERR, LABEL:]

Function	Statement Format
LIST FULL EXPRESSION	exp string = EXPLIS/name_string[,IFERR,LABEL:]
RENAME EXPRESSION	EXPRNM/name_string,TO,name_string [,IFERR,LABEL:]
SUBDIVIDE FACE	<u>obj list</u> = FACDIV/ent1,WITH,ent2[,CNT,c] [,IFERR,label:]
MOVE FACE	FACMOV/ent,TRIM,HEIGHT,h[,AXIS,i,j,k] [,IFERR,label:]
APPEND FILE TO CURRENT FILE	FAPEND/TXT,file#,'filespec'[,IFERR,label:]
CREATE NEW COMPONENT	<u>obj</u> = FCOMP/'filespec'[, 'component name'] [,REF,'reference set name'][,CSYS,csys] [,ORIGIN,pt], <u>objlist</u> [,IFERR,label:]
COPY FILE	FCOPY/'source filespec','destination filespec' [,IFERR,label:]
DELETE FILE	FDEL/'filespec'[,IFERR,label:]
RETRIEVE A FILE	FETCH/{PART,'filespec' TXT,file#,'filespec'} [,IFERR,label:]
MODIFY FILE HEADER	FHMOD/'filespec',[FNAME,'filename'] [,STATUS,status][,DESCR,'description'] [,CAREA,'customer area'] [,IFERR,label:]
READ FILE HEADER	FHREAD/'filespec'[,IFERR,label:]
FILE	FILE/{PART TXT[,file#][,LINNO]} [, 'filespec'][,IFERR,label:]
FILLET (Two Entities, Center Point)	<u>obj</u> = FILLET/ent1,ent2,CENTER,point,RADIUS,r [,NOTRIM][,IFERR,label:]
(Three Entities)	<u>obj</u> = FILLET/[{IN OUT TANTO}],ent1, [{IN OUT TANTO}],ent2,, [{IN OUT TANTO}],ent3, ,CENTER,point[,NOTRIM] [,IFERR,label:]



Function	Statement Format
(Two Entities, Positional Modifiers)	<code>obj = FILLET/"PMOD3",line1,"PMOD3",line2, RADIUS,r[,NOTRIM][,IFERR,label:]</code>
FILLET SURFACE	<code>obj = FILSRF/surf1,surf2,point[,TOLER,tl] ,RADIUS,r1[,r2] [LINEAR SSHAPE] [,lsurf1,lpoint1,lsurf2,lpoint2] [,VECT,x,y,z][,RESULT,result] [,IFERR,label:]</code>
MOVE FILE	<code>FMOVE/'source filespec','destination directory' [,{UPDATE NEWEST}] [,{VERIFY DELETE}] [,IFERR,label:]</code>
FORM-POSITIONAL TOLERANCE	<code>obj = FMPOS/(origin), {NONE' {EXT,{1 2 3 4}, line LEADER,[{LEFT RIGHT},] {obj1[,VIEW,'View Name] [, (origin)] (origin) } }[, TRIANG]} {,SYMBOL,number 'text' NEXTL} +</code>
FORMAT	<code>FNDSTR('object string','search string',pos)</code>
SET ENTITY FONT	<code>FONT/{SOLID DASH PHANTM CENTER}</code>
FONT DEFINITION	<code>string = FONTDF/ {INQUIR UGDFLT CUSTDF 'file spec'} [,IFERR,label:]</code>
LIST ALL	<code>FPRINT/file#[,LINNO][,USING,'image string']</code>
FOLDED RADIUS	<code>obj = FRDIM/(origin),arc,[VIEW,'View Name',] [,{ENDOF CENTER TANTO},,] "PMOD3" ,obj[,VIEW,'View Name'][,ANGLE,angle] ,x,y[,Dim. text][,APPEND,App. text]</code>
CONVERT REAL NO. TO CHARACTER STRING	<code>FSTR(n)</code>
CONVERT REAL NO. TO CHARACTER STRING (More Than 8 Chars)	<code>FSTRL(n)</code>



Function	Statement Format
CLOSE A FILE	FTERM/{PART[options] TXT,file#}[,IFERR,label:]
GENERAL CONIC (Five Points)	<u>obj</u> = GCONIC/ <u>point1,point2,point3,point4,point5</u>
(Four Points, One Slope)	<u>obj</u> = GCONIC/ <u>point1,point2,point3,point4</u> , VECT,x,y,z
(Three Points, Two Slopes)	<u>obj</u> = GCONIC/ <u>point1,point2,point3</u> ,VECT ,x1,y1,z1,x2,y2,z2
(Three Points, Anchor Point)	<u>obj</u> = GCONIC/ <u>point1,point2,point3</u> , ANCHOR, <u>point4</u>
(Two Points, Anchor Point, Rho Value)	<u>obj</u> = GCONIC/ <u>point1,point2</u> ,ANCHOR, <u>point3</u> ,rho
SIX COEFFECIENTS	<u>obj</u> = GCONIC/number list
GENERATE DRAFTING ENTITIES TO CURRENT SETTINGS	GENDIM/ <u>obj list</u>
GET LINE NUMBER	GETL(file#)
INDICATE GENERIC POINT	GPOS/'message',x-coord,y-coord,z-coord ,response
GRIP ARGUMENTS	GRARGS/ <u>parameter list</u> ,[,IFERR,label:]
UPDATE GRID DISPLAY	GRDDSP/[{ON OFF}]
DETECT UNDECLARED VARIABLES	GRIPSW/DECLRV
GROUP	<u>obj</u> = GROUP/ <u>obj list</u>
EDIT A GROUP	GRPEDT/{ADD{ <u>group entity</u> group name 'group name'} REMOVE} <u>entlist</u>
STOP PROGRAM EXECUTION	HALT
CROSSHATCHING	<u>obj</u> = HATCH/{XHATCH AFILL}, <u>obj list</u> [,VIEW,'View Name'][,IFERR,label:]



Statement Format Summary

Function	Statement Format
HOLE DIMENSION	<u>obj</u> = HDIM/(origin),arc[,VIEW,'View Name'] [,Dim. text][,APPEND,App. text]
HYPERBOLA	<u>obj</u> = HYPERB/point,semitransverse,semiconjugate ,dymin,dymax[,ATANGL,angle]
SELECT OBJECTS	IDENT/'message'[,SCOPE,{WORK ASSY REF}] , <u>obj list</u> [,CNT,count] [,CURSOR,x-coord,y-coord,z-coord] [,MEMBER,{ON OFF}],response
ID SYMBOL	<u>obj</u> = IDSYM/{CIR DCIRC SQR DSQR HEX DHEX TRIUP TRIDWN DATUM OBLNG}, (origin){NONE {ARROW DOT}} [,,{LEFT RIGHT}][,. <u>obj</u>] [,VIEW,'View Name'] [,,(origin)]},'text'[,,'text']
ARITHMETIC IF	IF/numerical expression,[label1:],[label2:],[label3:]
LOGICAL IF	IF/logical expression,statement
BLOCK IF	IFTHEN/e1 block1 [ELSEIF/e2 block2] . . . [ELSE block3] ENDIF
INITIALIZE DATA BASE CYCLING	INEXTE[/ALL]
INITIALIZE DATA BASE NON-GEOMETRIC ENTITY CYLCING	INEXTN/{type no. type GPA}[,subtype] [,IFERR,label:]
SET CROSSHATCHING PARAMETERS	INHAT/{XHATCH[,FNAME,'filename'][,UTIL], {material number 'material name'}, angle1,distance AFILL,fill#,angle2,scale} [,IFERR,label:]
INTERSECT SOLIDS	<u>obj list</u> = INTERS/ent,WITH,entlist[,CNT,c] [,IFERR,label:]

Function	Statement Format
INTEGER PORTION OF THE ARGUMENT	INTF(arg)
SOLIDS INTERFERENCE CHECKING	INTFER/ent, WITH, entlist, RESULT, numlist, IFERR, label:
INTERSECTION CURVE	<u>obj</u> = INTSEC/surf1, WITH, surf2 [, TOLER, tl] [, surf1, lpoint1, surf2, lpoint2[, VECT, x, y, z]] [, IFERR, label:]
ISOPARAMETRIC CURVE ON FACE	<u>obj</u> = ISOCR/obj{, UDIR VDIR}, num1 [, TOLER, num2][, CNT, num3][, IFERR, label:]
CONVERT INTEGER TO CHARACTER STRING	ISTR(n)
CONVERT INTEGER TO CHARACTER STRING (More Than 8 Chars)	ISTRL(n)
UNCONDITIONAL BRANCHING	JUMP/label:
CONDITIONAL BRANCHING	JUMP/label: +, [expression]
PROGRAM LABEL	LABEL:statement
LABEL	<u>obj</u> = LABEL/{LEFT RIGHT}, (origin), {obj[, VIEW, 'View Name']} [, (origin)] (origin)'text'[, 'text'] +
CREATE A LAYOUT (Single View)	LAYC/'layout name', 'view name' [, WORK[, {AUTO SCALE, s}]] [, IFERR, label:]
(Side by Side Views)	LAYC/'layout name', SIDE, 'view 1 name', 'view 2 name' [, WORK[, {AUTO SCALE, s}]] [, IFERR, label:]



Statement Format Summary

Function	Statement Format
(Top and Bottom Views)	LAYC/'layout name',TOP , 'view 1 name' , 'view 2 name' [,WORK[, {AUTO SCALE,s}]] [,IFERR,label:]
(Four Views)	LAYC/'layout name' , 'view 1 name' , 'view 2 name' , 'view 3 name' , 'view 4 name' [,WORK[, {AUTO SCALE,s}]] [,IFERR,label:]
(Six Views)	LAYC/'layout name' , 'view 1 name' , 'view 2 name' , 'view 3 name' , 'view 4 name' , 'view 5 name' , 'view 6 name' [,WORK[, {AUTO SCALE,s}]] [,IFERR,label:]
DELETE A LAYOUT	LAYD/'layout name' [,IFERR,label:]
EDIT A LAYOUT (Add a View)	LAYE/'layout name',] ADD,'view name',X1,Y1,X2,Y2 [,SAVE['new layout name']] [,IFERR,label:]
(Replace a View)	LAYE/'layout name',]REPL , 'old view name' , 'new view name' [,SAVE['new layout name']] [,IFERR,label:]
(Remove a View)	LAYE/'layout name',]REMOVE,'view name' [,SAVE['new layout name']] [,IFERR,label:]
(Save a layout)	LAYE/'layout name',]SAVE['new layout name'] [,IFERR,label:]

Function	Statement Format
LAYER CONTROL	LAYER/[WORK,n], [ACTIVE,{REST layer list[,CAT,'cat']}], [REF,{REST layer list[,CAT,'cat']}], [INACT,{REST layer list[,CAT,'cat']}]
RENAME A LAYOUT	LAYN/['old layout name',] 'new layout name' [,IFERR,label:]
RETRIEVE A LAYOUT	LAYR/'layout name'[, {AUTO SCALE,s}] [,IFERR,label:]
VERIFY LAYOUT	LAYV/['layout name',] variable list [,IFERR,label:]
DELETE A LINE IN A TEXT FILE	LDEL/file#[,START,start line#,END,end line#]
HORIZONTAL AND VERTICAL	<u>obj</u> = LDIM/{HORIZ VERT},(origin), [{ENDOF CENTER TANTO},] "PMOD3",obj1, [VIEW,'View Name',] [{ENDOF CENTER TANTO},] "PMOD3",obj2[,VIEW,'View Name'] [,Dim. text][,APPEND,App. text] [,OBLIQ, Angle]
PARALLEL	<u>obj</u> = LDIM/PARLEL,(origin), [{ENDOF CENTER TANTO},]"PMOD3", obj1[,VIEW,'View Name',] [{ENDOF CENTER TANTO},] "PMOD3",obj2[,VIEW,'View Name'] [,Dim. text][,APPEND,App. text]
PERPENDICULAR	<u>obj</u> = LDIM/PERP,(origin),baseline, [{ENDOF CENTER TANTO},]"PMOD3", obj[,VIEW,'View Name'][,Dim. text] [,APPEND,App. text]
NUMBER OF CHARS IN A STRING	LENF('string')
LINE (Parallel At A Dist)	<u>obj</u> = LINE/PARLEL, <u>line</u> ,"PMOD3",offset
(Parl/Perp To Line, TAngent To Curve)	<u>obj</u> = LINE/{PARLEL PERPTO}, <u>line</u> , { "PMOD3" <u>point</u> },TANTO, <u>curve</u>



Statement Format Summary

Function	Statement Format
(Thru A Pt, At An Angle)	<u>obj</u> = LINE/ <u>point</u> ,ATANGL,angle
(Point, Tangent To A Curve)	<u>obj</u> = LINE/ <u>point1</u> ,{LEFT RIGHT <u>point2</u> }, TANTO, <u>curve</u>
(Tangent To Two Curves)	<u>obj</u> = LINE/{LEFT RIGHT <u>point</u> },TANTO, <u>curve1</u> ,{LEFT RIGHT <u>point</u> },TANTO, <u>curve2</u>
(Thru A Point, Par/Perp To A Line)	<u>obj</u> = LINE/ <u>point</u> ,{PARLEL PERPTO}, <u>line</u>
(Thru A Point, Perp To A Curve)	<u>obj</u> = LINE/ <u>point1</u> , <u>point2</u> ,PERPTO, <u>curve</u>
(Between Two Existing Curves)	<u>obj</u> = LINE/ <u>point1</u> , <u>point2</u>
(Between Two Specified Points)	<u>obj</u> = LINE/x1,y1[,z1],x2,y2[,z2]
NATURAL LOGARITHM OF THE ARGUMENT	LOGF(arg)
LISTING DEVICE	LSTDEV/{CRT[,LPT OS] LPT OS NULL} [, 'filename'][,REPL]
REPOSITION A COORDINATE	pos = MAP/pos1,FROM, { <u>csys1</u> ,TO, <u>csys2</u> member_view_name,TO,parent_drawing_name}
CLASS SELECTION	MASK/{ALL NONE [OMIT,],ent type list}
TRANSLATE	matrix = MATRIX/TRANSL,dx,dy,dz
SCALE	matrix = MATRIX/SCALE{s ,xc,yc,zc}
ROTATE	matrix = MATRIX/{XYROT YZROT ZXROT},angle
MIRROR	matrix = MATRIX/MIRROR,{ <u>line</u> <u>plane</u> }
MULTIPLE TRANSFORMS (Concatenation)	matrix = MATRIX/matrix1,matrix2

Function	Statement Format
MAXIMUM VALUE IN THE ARGUMENT LIST	MAXF(arg[,arg] +)
CHOOSE MULTIPLE OPTIONS	MCHOOS/primary string , menu options , response array [,ALFACT,'message'] , response variable
WRITE TO MESSAGE MONITOR	MESSG/[TEMP,]string list
MINIMUM VALUE IN THE ARGUMENT LIST	MINF(arg[,arg] +)
REMAINDER OF THE DIVISION arg/mod	MODF(arg,mod)
REPLACE COLOR TABLE	NEWCTE/'filename'
CYCLE TO NEXT ENTITY	<u>obj</u> = NEXTE/IFEND,label:
CYCLE TO NEXT NON-GEOMETRIC ENTITY	string = NEXTN/IFEND,label1:[,IFERR,label2:]
NOTE	<u>obj</u> = NOTE/(origin), {scratch file #1 [,IFERR,label:] 'text' [, 'text'] + }
DECLARE NUMERICAL VARIABLE	NUMBER/name(dim1 [,dim2[,dim3]]) [,name(dim1 [,dim2[,dim3]])] +
ENTITY INFORMATION	OBTAIN/ <u>obj_list</u> , (variable list)
ORDINATE DIMENSION	<u>obj</u> = ODIM/(origin), {margin objid {HORIZ VERT} , origin objid} , [{ENDOF CENTER TANTO},] "PMOD3" , obj[,VIEW,'View Name'] [,DOGLEG,angle,distance][,Dim. text] [,APPEND,App. text][,IFERR,label:]



Function	Statement Format
OFFSET CURVES	<u>obj list</u> = OFFCRV/ <u>obj list</u> , {dist height, ang} , <u>ref point</u> [, STEP, n1] [{, EXT [, n2] , FILLET}] [, GROUP]
OFFSET BODY	<u>obj</u> = OFFSRF/ <u>ent</u> , dis [, TOLER, edge curve tolerance]
ORDINATE ORIGIN DIMENSION	<u>obj</u> = OODIM/[{ENDOF CENTER},] "PMOD3", <u>obj</u> [, VIEW, 'View Name'] [, QUAD, {quad_no}] [, ARROWS] [, NAME, 'name text'] [, SYMBOL, {symbol_no}] [, IFERR, label:]
ORDINATE MARGIN	<u>obj</u> = OOMGN/{HORIZ VERT}, <u>obj</u> [, VIEW, 'View Name',] {line [, VIEW, 'View Name'] xpos, ypos, xdir, ydir} [, offset distance] [, IFERR, label:]
PARABOLA	<u>obj</u> = PARABO/ <u>point</u> , focal length , dymin, dymax [, ATANGL, angle]
ENTER PARAMETERS	PARAM/'message' {, 'option' [, INT], variable} + [, ALTACT, 'message'], response
EXPAND A PATTERN	<u>obj</u> = PATEXP/ <u>ent</u> [, GROUP] [, LAYER] [, NOVIEW] [, PLMODS, value] [, IFERR, label:]
ASK/SET WORK, DISPLAYED, OR LOADED PARTS	string list = PARTOP/{ASK, {work dsplay all} SET, {work dsplay}, string} [, IFERR, LABEL:]
QUERY PART LOAD STATUS	num = PARTST/'part_name'
RETRIEVE A PATTERN	<u>obj</u> = PATRET/'file name' [, 'pattern name'] [, {matrix csys, scale}] [, AUTO] [, IFERR, label:]
UPDATE A PATTERN	PATUPD/ <u>ent</u> [, IFERR, label:]
LIST PATTERN RETRIEVAL ERRORS	PATLER/



Function	Statement Format
MODIFY PART FILE HEADER	PHMOD/['filename'][,STATUS,status] [,DESCR,'description'] [,CAREA,'customer area'] [,IFERR,label:]
READ PART FILE HEADER	PHREAD/['filename']{,STATUS[,status] ,DESCRnotes[,description] ,CAREA[,customer area] ,MCHFMT[,machine format] ,RELNO[,release num]} [,IFERR,label:]
PLANE (Plane Of An Arc Or Conic)	<u>obj</u> = PLANE/ <u>ent</u>
(Two Lines)	<u>obj</u> = PLANE/ <u>line1,line2</u>
(Parallel At a Distance)	<u>obj</u> = PLANE/PARLEL, <u>plane,point,d</u>
(Parallel To a Plane, Thru a Point)	<u>obj</u> = PLANE/PARLEL, <u>plane,THRU,point</u>
(Perpendicular To a Curve, At a Point)	<u>obj</u> = PLANE/PERPTO, <u>curve,THRU,point</u>
(Perpendicular To a Plane, Thru a Line)	<u>obj</u> = PLANE/PERPTO, <u>plane,THRU,line</u>
(Three Points)	<u>obj</u> = PLANE/ <u>point1,point2,point3</u>
(Principal Plane)	<u>obj</u> = PLANE/{XYPLAN[,Z-coord] YZPLAN[,X-coord] XZPLAN[,Y-coord]}[, <u>csys</u>]
PLACE SYMBOL	<u>obj</u> = PLCSYM/'symbol_name', <u>point,angle,</u> {SCALE,scale [,RATIO,ratio] SIZE,length,height} [,IFERR,label:]



Function	Statement Format
SET PARTS LIST MODE SETTINGS	PLMODE/sort field pos,sort mode,callout mode ,box mode,header mode ,update ID sym mode ,line space factor [,SECSRT,value1,value2] [,COLUMN,value1,value2,value3] [,RPMODE,value][,SKIPVL,value] [,FROZEN,value][,IFERR,label:]
CREATE OR REGEN. PARTS LIST NOTE	<u>obj</u> = PLNOTE/[xc,yc][,IFERR,label:]
PLOT	num = PLOT/{'plotter ID' DISPL},'drawing name' [,{COLOR LWIDTH DENS},pen list] [,JOB,'jobname'][,SCALE,s][,ANGLE,a] [,MEDIA,m][,COPIES,c][,ORIGIN,x,y] [,START,x,y][,PAUSE,'message'] [PLTTOL,t][,IFERR,label:]
SAVES/APPENDS A DRAWING TO A PLOT FILE	num = PLTSAV/'plotter ID','drawing name' [,{COLOR LWIDTH DENS},pen list] [,SCALE,s][,ANGLE,a] [,ORIGIN,x,y][,START,x,y] [PLTTOL,t][,IFERR,label:]
UPDATES THE PLOTTER DRAWING DEFAULTS	num = PLTUPD/'plotter ID','drawing name' [,{COLOR LWIDTH DENS},pen list] [,SCALE,s][,ANGLE,a][,MEDIA,m] [,COPIES,c][,ORIGIN,x,y][,START,x,y] [,IFERR,label:]
SUBMITS A PLOT FILE	num = PLTSUB/'plotter ID'[,JOB,'jobname'] [,MEDIA,m][,COPIES,c] [,PAUSE,'message'][,IFERR,label:]
DELETES PLOT FILE	num = PLTDEL/
LIST PARTS LIST	PLOUT/[IFERR,label:]
POINT (Center Of Circle)	<u>obj</u> = POINT/CENTER, <u>circle</u>
(Position On Arc)	<u>obj</u> = POINT/ <u>circle</u> ,ATANGL,angle
(End Point)	<u>obj</u> = POINT/ENDOF,"PMOD3", <u>ent</u>



Function	Statement Format
(Intersection Point)	<u>obj</u> = POINT/["PMOD2" <u>point</u>],INTOF, <u>ent1</u> , <u>ent2</u> [,IFERR,label:]
(Offset Point)	<u>obj</u> = POINT/ <u>point</u> ,DELTA,dx,dy,dz
(Polar Offset Point)	<u>obj</u> = POINT/ <u>point</u> ,POLAR,dist,angle
(Three Dimensional Offset Vector)	<u>obj</u> = POINT/ <u>point</u> ,VECT, <u>line</u> ,"PMOD3",dist
(Coordinates)	<u>obj</u> = POINT/x,y[,z]
(Pattern Point)	<u>obj</u> = POINT/x,y[,z],PATPNT
INDICATE SCREEN POSITION POINT	POS/'message',x-coord,y-coord,z-coord ,response
LIST	PRINT/[USING,'image string',]data list
PROJECT POINTS/CURVES	<u>obj list</u> = PROJ/ <u>obj list1</u> ,ON, <u>obj list2</u> [,TOLER,t] [,VECT,vect] [,ASSOC [,obja]] [,MOVE][,TRACRV]]
SUB ROUTINE HEADER	PROC[/dummy argument list]
ADD AN EXISTING PART AS A COMPONENT	<u>obj</u> = RCOMP/'filespec'[, 'component name'] [REF,'reference set name'] [, <u>csys</u>][,LAYER] [,IFERR,label:]
RADIUS	<u>obj</u> = RDIM/(origin),arc[,VIEW,'View Name'] [,Dim. text][,APPEND,App. text]
READ TEXT	READ/file#[,LINNO,line#][,USING,'image string'] [,IFEND,label:] [,IFERR,label:],variable list
COUNT REFERENCE SET MEMBERS	num = REFCNT/[PART,part_name, reference_set_name[,IFERR,label:]
CYCLE MEMBERS OF A REFERENCE SET	<u>obj</u> = REFMEM/[PART,'part_name',] 'reference_set_name' ,index[,IFERR,label:]



Statement Format Summary

Function	Statement Format
RETRIEVE EXPRESSION VALUE	REGF('expression name')
OBTAIN RELATIVE DISTANCE BETWEEN TWO ENTITIES	num list = RELDST/[MIN, ,ent1[,x1,y1[,z1]] ,ent2[,x2,y2[,z2]]
REMOVE ROW FROM PARTS LIST	REMOVPL/{kv1,kv2,kv3 obj_list} [,IFERR,label:]
RENAME SCRATCH FILE	RENAME/file#,'new filespec' [,IFERR,label:]
REVERSE FACE NORMALS	RENORM/obj_list
REPLACE CHARACTERS IN A STRING	REPSTR('object string' , 'search string' , 'replacement string', pos)
RESEQUENCE	RESEQ/file#[,START,line#,INCR,n]
RESET FILE POINTER	RESET/file#
RETURN TO MAIN PROGRAM	RETURN
SURFACE OF REVOLUTION	obj = REVSRF/ent,AXIS,line[,point][,start,end]
RULED SURFACE	obj = RLDSRF/ent1,[point1],ent2[,point2]
DISPLAY (REPAINT)	RPAIN
IMPORT AN EXISTING PART	RPATT/'filespec'[{,matrix ,csys,scale}] [,LAYER][,PLMODS,value][,NOVIEW] [,RETCAM,number][,IFERR,LABEL:]
IMPORT AN EXISTING PART GROUPED	obj = RPATTG/'filespec'[{,matrix ,csys,scale}] [,LAYER][,PLMODS,value][,NOVIEW] [,RETCAM,number][,IFERR,LABEL:]
REVERSE BLANK ALL	RVBLNK/[ALL]



Function	Statement Format
SYMBOL FONT FILE	<u>string</u> = SBFNT/{INQUIR UGDFLT CUSTDF 'file name'}[IFERR,label:]
SCALED VECTOR	SCALVF(scalar,vector)
CLOSE SUB-DIRECTORY	SCLOSE[/IFERR,label:]
SURFACE PARTIAL DERIVATIVE WITH RESPECT TO THE u PARAMETER	SDDUF(ent,u,v)
SURFACE PARTIAL DERIVATIVE WITH RESPECT TO THE v PARAMETER	SDDVF(ent,u,v)
CREATE SOLID SECTION	<u>obj list</u> = SECT/ <u>entlist</u> ,WITH, <u>ent</u> [,CNT,c] [,IFERR,label:]
SEW SHEETS TO CREATE BODY	<u>obj</u> = SEW/ <u>entlist</u> [,IFERR,label:]
SILHOUETTE CURVES	<u>obj</u> = SILHO/ <u>body</u> [,CNT,count][,IFERR,label:]
SIMPLIFY CURVES	<u>obj list</u> = SIMCRV/ <u>elist</u> [,BLANK ,DELETE][,TOLER,t][,CNT,c]
SINE OF angle	SINF(angle)
VECTOR NORMAL TO A SURFACE	SNORF(ent,u,v)
CREATE SOLID BLOCK	<u>obj</u> = SOLBLK/ORIGIN,xc,yc,zc ,SIZE,dx,dy,dz [,IFERR,label:]
SOLID BOX COORDINATES	<u>obj list</u> = SOLBOX/ <u>ent</u> [,IFERR,label:]



Function	Statement Format
CREATE SOLID CONE	<u>obj</u> = SOLCON/ORIGIN,xc,yc,zc ,HEIGHT,h ,DIAMTR,d1,d2 [,AXIS,i,j,k] [,IFERR,label:]
CUT SOLID	<u>obj list</u> = SOLCUT/ <u>obj list</u> ,WITH, <u>obj</u> [,CNT,c1[,c2]] [,IFERR,label:]
CREATE SOLID CYLINDER	<u>obj</u> = SOLCYL/ORIGIN,xc,yc,zc ,HEIGHT,h ,DIAMTR,d [,AXIS,i,j,k] [,IFERR,label:]
CREATE EDGE CURVES	<u>obj list</u> = SOLEDG/ <u>ent</u> [,CNT,c][,IFERR,label:]
RETURN EDGE OR FACE IDENTIFIERS	<u>obj list</u> = SOLENT/ <u>obj</u> {,FACE ,EDGE} {,ALL ,seqno}[,IFERR,label:]
CREATE EXTRUDED SOLID	<u>obj list</u> = SOLEXT/ <u>obj list</u> 1,HEIGHT,h[,AXIS,i,j,k] [,IFERR,label:]
CREATE SHADED SOLID IMAGE FILE	SOLPIX/'filename'[,QUICK][,SIZE,num,num] [,COLOR,num][,RESULT,numa] [,IFERR,label:] SOLPIX/'filename'[,SIZE,num,num] [,METHOD,num][,RESULT,numa] [,IFERR,label:]
CREATE SOLID PRISM	<u>obj</u> = SOLPRI/ORIGIN,xc,yc,zc, HEIGHT,j .DIAMTR,d ,SIDE,s [,AXIS,i,j,k] [,IFERR,label:]
CREATE SOLID OF REVOLUTION	<u>obj</u> = SOLREV/ <u>obj list</u> ,ORIGIN,xc,yc,zc ,ATANGL,a [,AXIS,i,j,k] [,IFERR,label:]



Function	Statement Format
CREATE SOLID SPHERE	<u>obj</u> = SOLSPH/ORIGIN, <u>xc</u> , <u>yc</u> , <u>zc</u> ,DIAMTR, <u>d</u> [,IFERR, <u>label</u> :]
CREATE SOLID TORUS	<u>obj</u> = SOLTOR/ORIGIN, <u>xc</u> , <u>yc</u> , <u>zc</u> ,RADIUS, <u>r1</u> , <u>r2</u> [,AXIS, <u>i</u> , <u>j</u> , <u>k</u>] [,IFERR, <u>label</u> :]
CREATE SOLID TUBE OR CABLE	<u>obj</u> = SOLTUB/ <u>obj list</u> ,DIAMTR, <u>d1</u> [, <u>d2</u>] [,TOLER, <u>t</u>] [,IFERR, <u>label</u> :]
OPEN SUB-DIRECTORY	SOPEN[/IFERR, <u>label</u> :]
SORT A FILE	SORT/from file#,to file#,'new filespec' [, {ASCEND DECEND}] ,start column,end column [{[, {ASCEND DECEND}] ,start column,end column} +] [,IFERR, <u>label</u> :]
U, V PARAMETERS OF A POINT ON A SURFACE	SPARF/ent,{point X,Y,Z}, <u>u</u> , <u>v</u>
POSITION ON A SURFACE	SPOSF(ent, <u>u</u> , <u>v</u>)
SPHERE (Arc)	<u>obj</u> = SPHERE/ <u>arc</u>
(Center, Radius)	<u>obj</u> = SPHERE/CENTER, <u>point1</u> ,RADIUS, <u>r</u> [, <u>plane</u> , <u>point2</u>]
(Tangent To Three Planes)	<u>obj</u> = SPHERE/TANTO, PLANE1, <u>plane2</u> , <u>plane3</u> ,CENTER, <u>point</u> ,RADIUS, <u>r</u>
SPLINE BY KNOT POINTS	<u>obj</u> = SPLINE/[CLOSED,] { <u>point</u> [, {VECT, <u>dx</u> , <u>dy</u> , <u>dz</u> TANTO, <u>curve</u> angle}]} +



Function	Statement Format
SPLINE BY APPROX. CURVES	<u>obj</u> = SPLINE/APPROX, [{BLANK DELETE}] [,TOLER,t] <u>obj list</u>
SPLIT SOLIDS	<u>obj list</u> = SPLIT/ <u>obj list1</u> ,WITH, <u>obj</u> [,CNT,c] [,IFERR,label:]
GEOMETRIC PROPERTIES OF A FACE AT A PARAMETER	SPROPF(<u>obj</u> ,u,v)
SQUARE ROOT OF arg	SQRTF(arg)
FREE FORM BODY (Swept Method)	<u>obj</u> = SSURF/ <u>ent1</u> , <u>ent2</u> ,p,c[, <u>ent3</u>]
FREE FORM BODY (Curve Mesh Method)	<u>obj</u> = SSURF/PRIMA, <u>obj list1</u> ,CROSS, <u>obj list2</u>
DECLARE STRING VARIABLE	STRING/name([dim1,[dim2]],n) [,name([dim1[,dim2]],n)]+
STORE VALUE IN AN EXPRESSION	STORE/'expression name',data
EXTRACT PORTION OF A CHARACTER STRING	SUBSTR('object string',pos,count)
SUBTRACT SOLIDS	<u>obj list</u> = SUBTRA/ <u>ent</u> ,WITH, <u>obj list1</u> [,CNT,C] [,IFERR,label:]
FIND SECTION LINE DATA	<u>obj list</u> = SXLDAT/ <u>object</u> , count[, REF, name] [, IFERR, label:]
FIND SECTION LINE SEGMENT DATA	<u>obj</u> = SXSEGD/ <u>object</u> [, RESULT, type] [, IFERR, label:]
TABULATED CYLINDER	<u>obj</u> = TABCYL/ <u>ent</u> ,x,y,z[,start,end]
ENTER TEXT	TEXT/'message',string-variable [,ALFACT,'message'] [,response[,DEFLT]
RETURN CURRENT TIME	TIME



Function	Statement Format
TRANSFORMATION	<u>obj_list</u> = TRANSF/matrix, <u>obj_list</u> [,MOVE][,TRACRV]
ENTITY TYPE	TYPF(<u>ent</u>)
DEFINE SYMBOL	UDFSYM/'symbol_name',['filespec',] point1,point2, <u>obj_list</u> [,factor] [,IFERR,label:]
USER FUNCTION ARGUMENTS	UFARGS/ <u>parameter list</u> [,IFERR,label:]
ENCODE UG/MANAGER PART FILE NAME	string = UGMGRE / PRTNUM,string, PRTREV,string [,PRTTYP,string] [,PRTFIL,string] [,IFERR,label:]
DECODE UG/MANAGER PART FILE NAME	string_list = UGMGRD / string [,IFERR,label:]
UNBLANK	UNBLNK/{ <u>obj_list</u> ALL}
UNGROUP	UNGRP/[TOP,] <u>obj_list</u>
UNITE SOLIDS	<u>obj_list</u> = UNITE/ <u>obj</u> ,WITH, <u>obj_list1</u> [,CNT,c] [,IFERR,label:]
WHERE USED REPORT	USED/['component name'] [,ROOT, 'dir spec'] [,IFERR,label:]
UNIT VECTOR	UNITF(A)
UPDATE COMPONENTS	<u>obj</u> = UPDATE/component[,ROOT, 'directory'] [,REPORT] [, 'reference set name'] [,PART, 'part'] [,IFERR,label:]
UPGRADE COMPONENTS	<u>obj</u> = UPGRAD/{ALL COMP,component list} [,RECURS] [,CREATE] [,STATUS,status] [,IFERR,label:]
CONVERT STRING TO A REAL NUMBER	VALF('string')



Statement Format Summary

Function	Statement Format
RETURN VIEW DEPENDENT LOCATION OF A POINT ON A CURVE	num = VDCPRM/ <u>ent</u> ,{point x,y,z}[,'view name']
ERASE FROM VIEW	VEDIT/ <u>obj list</u> ,ERASE[,'view name']
MODIFY CURVE SEGMENTS	VEDIT/ <u>obj list</u> [, <u>param1</u> , <u>param2</u>] [,{COLOR,c FONTN,'fname' FONT,f DENS,d LWIDTH,w}] [,'view name']
REMOVE SELECTED VIEW MODS	VEDIT/ <u>obj list</u> ,REMOVE[,'view name']
REMOVE ALL VIEW MODS	VEDIT/ALL,REMOVE[,'view name']
CONVERT SELECTED VIEW DEP. TO MODEL	VEDIT/ <u>obj list</u> ,MODEL[,'view name']
CONVERT ALL VIEW DEP. TO MODEL	VEDIT/ALL,MODEL[,'view name']
CONVERT MODEL TO VIEW DEPENDENT	VEDIT/ <u>obj list</u> ,VIEW[,'view name']
CHANGE VIEW	VIEW/n
CREATE A VIEW	VIEWC/'view name', { <u>csys</u> 'base view name' [, <u>VMODS</u>][, <u>VDEP</u>]} [, <u>WORK</u> [,{ <u>AUTO</u> <u>SCALE</u> ,s}]] [, <u>IFERR</u> ,label:]
DELETE A VIEW	VIEWD/'view name' [, <u>IFERR</u> ,label:]



Function	Statement Format
EDIT VIEW	<pre>VIEWE/['view name'] [{CSYS,csys matrix}] [{[,SCALE,s1][,CENTER,x,y[,z]] ,AUTO}], {PARLEL PERSP,d1 EYEPT,x,y[,z]}] [{[,FRONTZ,{z1 OFF}] [,BACKZ,{z2 OFF}] ,AUTOZ}] [,REF,x,y[,z]] [,DSCALE,s2] [,SRFDSP,n] [,DCUE[,OFF]] [,SAVE[, 'new view name'] [,VMODS][,VDEP]]] [,BLEND,{VSBL INVSBL}] [,SMOOTH,{VSBL INVSBL}] [,SILHO,{VSBL INVSBL}] [,HIDDEN,{VSBL INVSBL DASH}] [,IFERR,label:]</pre>
VIEW LAYER VISIBILITY MASK	<pre>VIEWLC/['view name'] {,RESET [,VSBL{,REST [,layer list] [,CAT,cat list]}][,INVSBL{,REST [,layer list][,CAT,cat list]}]} [,IFERR,label:]</pre>
RENAME A VIEW	<pre>VIEWN/['old view name',] 'new view name' [,IFERR,label:]</pre>
VERIFY VIEW	<pre>VIEWV/['view name',] variable list[,IFERR,label:]</pre>
LENGTH OF A VECTOR (Magnitude)	<pre>VLENF(A)</pre>
WRITE TEXT TO A TEXT FILE	<pre>WRITE/file#[,LINNO,line#][,USING,'image string'], data list</pre>
EXECUTE OPERATING SYSTEM FUNCTIONS (Format 1)	<pre>XSPAWN/[CONCUR,][PROG,]'program name' [, 'argument list1', ..., 'argument listn'] [,RESULT,result string variable] [,IFERR,label:]</pre>
(Format 2)	<pre>XSPAWN/UFUN,'program name'[,IFERR,label:]</pre>



(This Page Intentionally Left Blank)



GPA Symbol Format Summary

Appendix E

On the following pages you will find an alphabetical list of the GPAs, their purpose, access type, data type, and range. The following are explanations of letters and/or characters which will be found in the *RANGE* column.

LT	--	Less Than
LE	--	Less Than Or Equal To
GT	--	Greater Than
GE	--	Greater Than Or Equal To
**	--	Any valid Unigraphics numerical value

The following are explanations of letters and/or characters which will be found in the *ACCESS* and *DATA TYPE* columns.

N	--	Number Valued
S	--	String Valued
E	--	Entity Valued
RW	--	Read/Write
RO	--	Read Only
C	--	Constant (value which may not be changed)

NOTE GPAs that access the file header to read data, must be preceded by the **FHREAD** or **DOPEN** and **DNEXT** statements. These statements get the information from the file header, and store it where the GPAs have access to it. An asterisk (*) will appear next to all GPAs which require the use of **FHREAD** or **DOPEN** and **DNEXT**.

Function	Symbol	Access Type	Data Type	Range
ACCESS ABSOLUTE COORDINATE SYSTEM	&ABS	RO	E	
ACTIVE PART STATUS No Active Part Active Part	&ACTPRT	RO	N	[1,2] =1 =2



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
ALL AROUND SYMBOL DISPLAY Displayed Not displayed	&ALARND &YES &NO	RW C C	N C C	[1,2] =1 =2
MODELING ANGLE TOLERANCE	&ANGTOL	RW	N	GT 0
APPENDED TEXT SITE Below After Above Before Before and After	&APSITE &BELOW &AFTER &ABOVE &BEFORE &BEFAFT	RW C C C C C	N N N N N N	[1..5] =1 =2 =3 =4 =5
RADIUS DIMENSION DISPLAY Leader drawn to arc center Leader extends to text box	&ARCCTR &YES &NO	RW C C	N N N	[1,2] =1 =2
ARROW DISPLAY Two Arrows First Arrow Second Arrow No Arrows	&ARDSP &BOTH &FIRST &SECOND &NONE	RW C C C C	N N N N N	[1..4] =1 =2 =3 =4
ARROWHEAD FILL DISTANCE	&ARFLDT	RW	N	GT 0
ARROWHEAD INCLUDED ANGLE	&ARIANG	RW	N	GT 0 LT 180
ARC LENGTH SYMBOL Symbol displayed Symbol not display	&ARLNSM &YES &NO	RW C C	N C C	[1,2] =1 =2
ARROWHEAD 1 TYPE Closed Open Cross Dot Origin Symbol No arrowhead	&ARROW &CLOSED &OPEN &ARCHCR &DOT &ORGSYM &NOARHD	RW C C C C C C	N N N N N N N	[1..6] =1 =2 =3 =4 =5 =6



Function	Symbol	Access Type	Data Type	Range
ARROWHEAD 2 TYPE Closed Open Cross Dot Origin Symbol No arrowhead	&ARROW2 &CLOSED &OPEN &ARCHCR &DOT &ORGSYM &NOARHD	RW C C C C C C	N N N N N N N	[1..6] =1 =2 =3 =4 =5 =6
ARROWHEAD SIZE	&ASIZE	RW	N	GT 0
ASPECT RATIO	&ASPECT	RW	N	GT 0
ATTRIBUTE NAME DISPLAY Display Do Not Display	&ATTDIS &YES &NO	RW C C	N N N	[1,2] =1 =2
ATTRIBUTE TYPE Integer Floating Point Date/Time Null Variable Length String (132) Indicates All Data Types Reference	&ATTYPE &INT &REAL &DATE &NULL &STRING &ALLTYP &REFRNC	RW C C C C C C C	N N N N N N N N	[1..7] =1 =2 =3 =4 =5 =6 =7
ANGULAR DIMENSION UNITS Decimal Fraction Whole Deg's. Deg's./Min's. Deg's./Min's./Sec's	&AUNIT &DECIM &DEG &DEGM &DEGMS	RW C C C C	N N N N N	[1..4] =1 =2 =3 =4
CURRENT EXECUTION MODE Batch Mode Interactive Mode	&BATCH	RO	N	[1,2] =1 =2
CUSTOMER AREA OF THE FILE HEADER	&CAREA *	RW	S	124 chars.
BACKGROUND COLOR	&BGCLR	RW	N	[0.0–1.0]
CHARACTER FONT	&CFONT	RW	N	GT 0



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
CHAINING METHOD Simple XY Plane Only XY Plane To The Left XY Plane To The Right	&CHAIN &SIMPLE &CWCS &CWCSL &CWCSR	RW C C C C	N N N N N	[1..4] =1 =2 =3 =4
CHAINING TOLERANCE	&CHTOL	RW	N	[0..100]
CLIPPING TOLERANCE	&CLPTOL	RW	N	[.001 – .10]
COMPOSITE FRAME DISPLAY Displayed Not displayed	&CMPFRM &YES &NO	RW 	N C C	[1,2] =1 =2
CREATION MODE View Independent View Dependent	&CNMODE &MODEL &VIEW	RW C C	N N N	[1,2] =1 =2
CREATION DATE OF THE FILE DD–MM–YY	&CRDATE*	RO	S	8 chars.
CREATION TIME OF THE FILE	&CRTIME *	RO	S	5 chars.
CHARACTER SIZE	&CSIZE	RW	N	GT 0
CHARACTER SLANT IN DEG'S.	&CSLANT	RW	N	±45°
NEW COORDINATE SYSTEM STATUS	&CSMODE &NORMAL &TEMP	RW C C	N N N	[1,2] =1 =2
NAME OF WORKING DIRECTORY	&CURDIR	RW	S	99 chars.
CURRENT DRAWING	&CURDRW	RW	S	30 chars.
CURRENT LAYOUT	&CURLAY	RW	S	30 chars.
DASH SIZE	&DASHSZ	RW	N	GTE 0
DIMENSION PLACES PAST THE DECIMAL	&DDECPL	RW	N	[0..7]
DUAL DIMENSION PLACES PAST THE DECIMAL	&DDDECP	RW	N	[0..7]



Function	Symbol	Access Type	Data Type	Range
SETS THE NUMBER OF DECIMAL PLACES FOR NUMERICAL DISPLAYS	&DECPL	RW	N	[0..9]
DECIMAL POINT CHARACTER Period Comma	&DECPT &PERIOD &COMMA	RW C C	N N N	[1,2] =1 =2
COMMON FRACTION DENOMINATOR	&DENOM	RW	N	[1,2,4,8,16,32,64]
DUAL FRACTION DENOMINATOR	&DENOMD	RW	N	[1,2,4,8,16,32,64]
DESCRIPTION AREA OF THE FILE HEADER	&DESCR *	RW	S	132 chars.
DIAMETER SYMBOL	&DIASYM	RW	S	6 char's.**
NAME OF LAST DIRECTORY OPENED BY DOPEN OR SOPEN	&DIR *	RO	S	10 chars.
PATHNAME OF LAST DIRECTORY OPENED BY DOPEN OR SOPEN	&DIRPTH *	RO	S	99 chars.
DISTANCE FROM DIM. LINE TO TEXT	&DISTDT	RW	N	GT 0
MODELING DISTANCE TOLERANCE	&DISTOL	RW	N	GT0
EXTENSION LINE DISTANCE TO REFERENCE POINT	&DISTXR	RW	N	GT 0
EXTENSION LINE DISTANCE PAST DIMENSION LINE	&DISTXD	RW	N	GT 0
LINE WIDTH (for dimensions) Normal Thick Thin	&DLWID &NORMAL &THICK &THIN	RW C C C	N N N N	[1..3] =1 =2 =3
OPTIONAL \$ DIRECTORY	&DOLLIB	RW	S	99 chars.
DIAMETER OF DOT ARROWHEAD	&DOTDIA	RW	N	GT 0



Function	Symbol	Access Type	Data Type	Range
LN, AR, CSIZE RELATIONSHIP Arrowhead & Line Sizes Related To Char. Size	&DPREL &YES	RW C	N N	[1,2] =1
Arrowhead & Line Sizes Not Related To Char. Size	&NO	C	N	=2
DIAMETER AND RADIUS SYMBOL SITE Below Dim. & Tol. Text	&DRSITE	RW	N	[1..5]
After Dim. & Tol. Text	&BELOW	C	N	=1
Above Dim. & Tol. Text	&AFTER	C	N	=2
Before Dim. & Tol. Text	&ABOVE	C	N	=3
Omit Symbol	&BEFORE &OMIT	C C	N N	=4 =5
CURRENT DRAWING STATE Modeling Display	&DSTATE	RW C	N N	[1,2] =1
Drawing Display		C	N	=2
EXTENSION LINE DISTANCE TO SECOND REFERENCE POINT	&DSTXR2	RW	N	GT 0
DUAL TOLERANCE PLACES PAST THE DECIMAL	&DTDECP	RW	N	[0..7]
CURVE AND SHEET BODY DISPLAY TOLERANCE	&DTOLC	RW	N	GT 0
CURVE AND SHEET BODY DISPLAY TOLERANCE	&DTOLS	RW	N	GT 0
DUAL FORMAT Dual Below	&DUAL &BELOW	RW C	N N	[1..5] =1
Dual After	&AFTER	C	N	=2
Dual Above	&ABOVE	C	N	=3
Dual Before	&BEFORE	C	N	=4
Non-dual	&NODUAL	C	N	=5
DUAL LINEAR DIMENSION UNITS Millimeters	&DUNITD &MM	RW C	N N	[1..5] =1
Meters	&M	C	N	=2
Inches	&INCH	C	N	=3
Arch. Feet & Inches	&ARCHFI	C	N	=4
Eng. Feet & Inches	&ENGFI	C	N	=5



Function	Symbol	Access Type	Data Type	Range
LINEAR DIMENSION UNITS Millimeters Meters Inches Arch. Feet & Inches Eng. Feet & Inches	&DUNIT &MM &M &INCH &ARCHFI &ENGFI	RW C C C C C	N N N N N N	[1..5] =1 =2 =3 =4 =5
CAM: LIST DUPLICATE MERGES List Do Not List	&DUPCAM &YES &NO	RW C C	N N N	[1,2] =1 =2
OBJECT SITE RELATED TO TEXT Top Left Top Center Top Right Mid-left Mid-center Mid-right Bottom Left Bottom Center Bottom Right	&ENSITE &TOPL &TOPC &TOPR &MIDL &MIDC &MIDR &BOTL &BOTC &BOTR	RW C C C C C C C C	N N N N N N N N N	[1..9] =1 =2 =3 =4 =5 =6 =7 =8 =9
OBJECT COLOR Blue Green Cyan Red Magenta Yellow White Olive Pink Brown Orange Purple Dark Red Aquamarine Gray	&ENTCLR &BLUE &GREEN &CYAN &RED &MAGENT &YELLOW &WHITE &OLIVE &PINK &BROWN &ORANGE &PURPLE &DKRED &AQUAMR &GRAY	RW C C C C C C C C C C C C C C	N N N N N N N N N N N N N N N	[1..15] =1 =2 =3 =4 =5 =6 =7 =8 =9 =10 =11 =12 =13 =14 =15
PROCESS CONTROL IF ERRORS EXIST Terminate Processing Continue Processing	&ERRMOD &TERM &CONTIN	RW C C	N N N	[1,2] =1 =2



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
RETURN LAST ERROR NUMBER	&ERROR	RO	N	Any Valid Error Number
OPTIONAL ! DIRECTORY	&EXCLIB	RW	S	99 chars.
EXPAND or UNEXPAND VIEW	&EXPAND	RW	S	30 chars.
FILLED ARROWHEAD DISPLAY Display Do Not Display	&FLARDP &YES &NO	RW C C	N N N	[1,2] =1 =2
FAST FONT DISPLAY Display Do Not Display	&FASTF &YES &NO	RW C C	N N N	[1,2] =1 =2
ARROWHEAD FILL CONTROL Display Do Not Display	&FILARW &YES &NO	RW C C	N N N	[1,2] =1 =2
FILL PATTERN TYPE Cork/Felt/Fiber Sound Insulation Concrete Earth Rock Sand Liquids Wood - Across Grain Wood - With Grain Solid Fill	&CORK &SNDINS &CONCRT &EARTH &ROCK &SAND &LIQUID &AGRAIN &WGRAIN &SOLIDF	C C C C C C C C C C C	N N N N N N N N N N N	[1..10] =1 =2 =3 =4 =5 =6 =7 =8 =9 =10
LENGTH OF CURRENT FILE	&FLEN *	RO	N	GE 0
FRAME SIZE OF F&PT SYMBOL	&FMSIZE	RW	N	GT 0
NAME OF CURRENT FILE	&FNAME *	RW	S	30 chars.
LINE FONT MODE Hardware Line Fonts Software Line Fonts	&FMODE	RW	N	[1,2] =1 =2



Function	Symbol	Access Type	Data Type	Range
LINE FONT Solid Dashed Phantom Centerline Dotted Long Dashed Dotted Dashed	&FONT &SOLID &DASHED &PHANTM &CLINE	RW C C C C	N N N N N	[1..7] =1 =2 =3 =4 =5 =6 =7
LINE FONT NAME	&FONTN	RW	S	30 chars.



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
FORMAT OF CURRENT FILE Binary CL File Post Binary File UNIPCB Board Computer Graphics Metafile Display File ASCII Crosshatch File CL-File (APT Format) Customer Defined Miscellaneous Data File Directory Display File Display File NC Facet File Parts List Format GRIP Batch Intermediate GRIP Intermediate GRIP Batch Execution GRIP Execution UNISOLIDS Image File ASCII CL file Native Lang. Binary Post Machine Data NC TV File Post Processing Data File Unigraphics Part Patran Neutral Binary UNIPCB Part Pattern Facets UGSOLID Pixel File UNIPCB Package Symbol Definition Stereo Lithography Text File Aldus TIFF User Analysis Definitions User Analysis Master User Defined Feature UGI File UNISOLIDS Checkpoint NASTRAN Data File Misc. (Undefined)	&FORMAT*	RO	S	4 chars. BCL BIN BRD CGM CGP CHX CLF CUST DATA DIR DISP DSS FCET FMT GRII GRIU GRXI GRXU IMAG ISO LANG MDF NCTV PAB PART PBNF PCPT PFAC PIXL PKG SBF STL TEXT TIFF UAI UAM UDF UGI USCK UT1 (4 Blanks)
FILE STATUS OF CURRENT FILE	&FSTAT *	RW	N	Any



Function	Symbol	Access Type	Data Type	Range
CURRENT FILE SYSTEM Native File System	&FSYS	RO	N N	[2] =2
SPACE FACTOR	&GAPFAC	RW	N	GT 0
GRID BOUNDS-POLAR GRIDS	&GBNDSP	RW	N	[1..4]
GRID BOUNDS-RECT. GRIDS	&GBNDSR	RW	N	[1..4]
BATCH PARAMETER STRING	&GBSTR	RO	S	132 chars.
GRID SIZE	&GSIZE	RW	N	GT 0
ANGULAR SPACING—POLAR GRIDS	&GSIZEA	RW	N	GT 0
RADIAL SPACING—POLAR GRIDS	&GSIZER	RW	N	GT 0
X—SPACING—RECTANGULAR GRIDS	&GSIZEX	RW	N	GT 0
Y—SPACING—RECTANGULAR GRIDS	&GSIZEY	RW	N	GT 0
GRID SNAP Snap Enabled Snap Disabled	&GSNAP &YES &NO	RW C C	N N N	[1,2] =1 =2
GRID TYPE Rectangular Grid Polar Grid	>YPE &RECT &POLAR	RW C C	N N N	[1,2] =1 =2
CROSSHATCH ANGLE	&HANGLE	RW	N	±180°
CROSSHATCH DISTANCE	&HDIST	RW	N	GT 0
CROSSHATCH MATERIAL TYPE Iron/General Use Steel Brass,Copper Rubber or Plastic Refractory Marble, Slate or Glass Lead Aluminum, Magnesium Electrical Winding Thermal Insulation	&HMAT &IRON &STEEL &BRASS &RUBBER &REFRAC &MARBLE &LEAD &ALUM	RW C C C C C C C C	N N N N N N N N N N	[1..10] =1 =2 =3 =4 =5 =6 =7 =8 =9 =10



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
CROSSHATCH TOLERANCE	&HTOLER	RW	N	GT 0
ID–SYMBOL SIZE	&IDSIZE	RW	N	GT 0
FRACTION DISPLAY TYPE Decimal 2/3 Size Common 3/4 Size Common Full Size Common	&FTYPE &DECIM &FRAC &TQFRAC &FSFRAC	RW C C C C	N N N N N	[1..4] =1 =2 =3 =4
DUAL FRACTION DISPLAY TYPE Decimal 2/3 Size Common 3/4 Size Common Full Size Common	&FTYPED &DECIM &FRAC &TQFRAC &FSFRAC	RW C C C C	N N N N N	[1..4] =1 =2 =3 =4
OBJECT SITE INDICATOR Display Do Not Display	&INDIC &YES &NO	RW C C	N N N	[1,2] =1 =2
LAST ACCESSED DATE OF CURRENT FILE	&LADATE *	RO	S	8 chars.
RETURN VALUE OF NATIVE LANGUAGE MENU NAME STRING	&LANG	RO	S	16 chars.
LAST ACCESSED TIME OF CURRENT FILE	&LATIME *	RO	S	5 chars.
LOCAL CHECKING OF SOLIDS Local Checking Off Local Checking On	&LCHECK &OFF &ON	RW C C	N N N	[0..1] =0 =1
LISTING DEVICE CRT Line Printer Remote Line Printer OS Device CRT & Line Printer CRT & Remote Line Printer CRT & OS Device Null	&LDEV &CRT &LP	RO C C	N N N	[1..10] =1 =2 =3 =5 =6 =7 =9 =10



Function	Symbol	Access Type	Data Type	Range
LEADER ORIENTATION Left Right	&LEADER &LEFT &RIGHT	RW C C	N N N	[1..3] =1 =3
LINE FACTOR	&LINFAC	RW	N	GT 0
LAST MODIFIED DATE OF CURRENT FILE	&LMDATE *	RO	S	8 chars.
LAST MODIFIED TIME OF CURRENT FILE	&LMTIME *	RO	S	5 chars.
LINE BETWEEN ARROWS Display Do not display	&LNARRW &YES &NO	RW C C	N N N	[1,2] =1 =2
CURRENT LINE BEING EXECUTED IN A GRIP PROGRAM	&LNUM	RO	N	GT 0
LINE WIDTH (for geometry) Normal Thick Thin	&LWIDTH &NORMAL &THICK &THIN	RW C C C	N N N N	[1..3] =1 =2 =3
MACHINE FORMAT OF CURRENT FILE Data General Digital AIX, UNIX AXP, OSF, Ultrix	&MCHFMT*	RO	S	GE 3 chars. DG DEC UNIX ULT
MEMBER VIEW DISPLAY Complete Outline Only	&MVDISP &YES &NO	RW C C	N N N	[1,2] =1 =2
OWNER OF CURRENT FILE	&OWNER*	RW	S	10 chars.
PART OBJECT IDENTIFIER	&PARATT	RO	E	
PATTERN MAX/MIN BOX DISPLAY STATUS Display Do not display	&PATBOX &YES &NO	RW C C	N N N	[1,2] =1 =2



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
PATTERN CONTROL POINT DISPLAY STATUS Display Do not display	&PATCPT &YES &NO	RW C C	N N N	[1,2] =1 =2
PATTERN ERRORS Pattern errors occurred No pattern errors occurred	&PATERR &YES &NO	RO C C	N N N	[1,2] =1 =2
PATTERN DATA ARCHIVING Save pattern data Do not save pattern data	&PATFIL &YES &NO	RW C C	N N N	[1,2] =1 =2
PATTERN ORIGIN MARKER DISPLAY STATUS Display Do not display	&PATMRK &YES &NO	RW C C	N N N	[1,2] =1 =2
PATTERN SHADED DATA ARCHIVING Save pattern shaded data Do not save pattern shaded data	&PATSHD &YES &NO	RW C C	N N N	[1,2] =1 =2
PROTECTION CLASS OF CURRENT FILE	&PCLASS *	RW	S	8 chars.
POSITION INDICATOR TYPE Screen Cursor Data Tablet	&PITYPE &CURSOR &TABLET	RW C C	N N N	[0,1] =0 =1



Function	Symbol	Access Type	Data Type	Range
CURRENT PLATFORM Apollo VAX/VMS HP CISC HP RISC SUN 3 SUN SPARC DEC RISC (ULTRIX) SGI Data General (DG) IBM MVS IBM AIX AXP/OSF AXP/VMS AXP Windows NT™ Intel x86 Windows NT™	&PLTFRM	RO	N	[1..15] =1 =2 =3 =4 =5 =6 =7 =8 =9 =10 =11 =12 =13 =14 =15
CURRENT PART NAME	&PNAME	RW	S	30 chars.
CURRENT FILE SPECIFICATION	&PSPEC	RW	S	132 chars.
RADIUS SYMBOL	&RADSYM	RW	S	6 char's. **
EXPRESSIONS A-L Access Expression "A" Access Expression "B" Access Expression "C" Access Expression "D" Access Expression "E" Access Expression "F" Access Expression "G" Access Expression "H" Access Expression "I" Access Expression "J" Access Expression "K" Access Expression "L" Other Expressions	®A ®B ®C ®D ®E ®F ®G ®H ®I ®J ®K ®L ®('exp name')	RW RW RW RW RW RW RW RW RW RW RW RW RW	N N N N N N N N N N N N N	* * * * * * * * * * * * *
*Any real number				



Function	Symbol	Access Type	Data Type	Range
RELEASE NUMBER OF CURRENT FILE	&RELNO *	RO	S	GE 7 chars. V1 V2 V3 V4 V41 V5 V6 V7 V8 V8A V9 V9M V91 V10 V10.1 V10.2 V10.3 V10.3.3 V10.4
RETAIN ANNOTATIONS	&RETANN	RW	N	[0,1]
RUNOUT FILLED ARROWHEAD DISPLAY Displayed Not displayed	&RNFILL &YES &NO	RW	N C C	[1,2] =1 =2
RUNOUT ARROWHEAD FILLED DISTANCE	&RNDIST	RW	N	GT 0
SYMBOL DISPLAY MODE IN LINE FONT Complete Outline	&SBDISP &YES &NO	RW C C	N N N	[1,2] =1 =2
SOLID DENSITY	&SDENS	RW	N	GE 0
SOLID DENSITY UNITS Pounds Per Cubic Inch Pounds Per Cubic Foot Grams Per Cubic Centimeter Kilograms Per Cubic Meter	&SDUNIT &LBIN &LBFT &GCM &KGM	RW C C C C	N N N N N	[1..4] =1 =2 =3 =4



Function	Symbol	Access Type	Data Type	Range
NUMBER OF SURFACE GRIDS IN U DIRECTION	&SGRIDU	RW	N	GE 0
NUMBER OF SURFACE GRIDS 4 IN V DIRECTION	&SGRIDV	RW	N	GE 0
SPACE SIZE	&SPCSZ	RW	N	GE 0
SPECKLE TOLERANCE	&SPKTOL	RW	N	[.001 – .10]
STUBB SIZE	&STUBB	RW	N	GE 0
TRAILING ZERO SUPPRESSION Display Do not display	&SUPRES &YES &NO	RW C C	N N N	[1,2] =1 =2
SYMBOL SIZE	&SYMBSZ	RW	N	GT 0
DISTANCE BETWEEN TEXT AND DIAMETER SYMBOL	&SYMDIS	RW	N	GE 0
SYSTEM COLOR Blue Green Cyan Red Magenta Yellow White Olive Pink Brown Orange Purple Dark Red Aquamarine Gray	&SYSCLR &BLUE &GREEN &CYAN &RED &MAGENT &YELLOW &WHITE &OLIVE &PINK &BROWN &ORANGE &PURPLE &DKRED &AQUAMR &GRAY	RW C C C C C C C C C C C C C C	N N N N N N N N N N N N N N N	[1..15] =1 =2 =3 =4 =5 =6 =7 =8 =9 =10 =11 =12 =13 =14 =15
TOLERANCE PLACES PAST THE DECIMAL	&TDECPL	RW	N	[0..7]
TEXT BOX DISPLAY Display Do not display	&TEXBOX &YES &NO	RW C C	N N N	[1,2] =1 =2



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
TEXT ORIENTATION Horizontal Aligned With Dim. Lines Over Solid Dim. Lines By angle Perpendicular	&TEXTOR &TXHOR &TXALIN &TXOVER &TXBANG &TXPERP	RW C C C C C	N N N N N N	[1..5] =1 =2 =3 =4 =5
TEXT LINE WIDTH Normal Thick Thin	&TLWID &NORMAL &THICK &THIN	RW C C C	N N N N	[1..3] =1 =2 =3
TOLERANCE TYPE None Limit Single Line Limit Double Line (max/min tolerance) Bilateral Single Line (plus or minus) Bilateral Double Line (upper, lower limit) Unilateral Above Unilateral Below Basic Reference Limit, Larger First Limit, Larger Below	&TOLER &NOTOL &LIMSA &LIMDA &BILS &BILD &UNILA &UNILB &BASIC &REF &LIMSB &LIMDB	RW C C C C C C C C C C C	N N N N N N N N N N N N	[1..11] =1 =2 =3 =4 =5 =6 =7 =8 =9 =10 =11
TOLERANCE VALUE (English): LOWER	&TOLOW	RW	N	± Infinity
TOLERANCE VALUE (Metric): LOWER	&TOLOWM	RW	N	±Infinity
TOLERANCE VALUE (English): UPPER	&TOLUP	RW	N	±Infinity
TOLERANCE VALUE(Metric): UPPER	&TOLUPM	RW	N	±Infinity
TOLERANCE SITE Below Dimension Text After Dimension Text Above Dimension Text	&TOSITE &BELOW &AFTER &ABOVE	RW C C C	N N N N	[1..3] =1 =2 =3
TEXT ANGLE	&TXANGL	RW	N	±360°



Function	Symbol	Access Type	Data Type	Range
TEXT AND ARROW LOCATION Automatic Manual Loc., Arrows In Manual Loc., Arrows Out	&TXARR &AUTO &ARRIN &ARROUT	RW C C C	N N N N	[1..3] =1 =2 =3
TEXT JUSTIFICATION Left Center Right	&TXJUST &LEFT &CENTER &RIGHT	RW C C C	N N N N	[1..3] =1 =2 =3
READ USER EXIT NUMBER	&UENUM	RO	N	[1..58]
ACCESS USER EXIT STRING	&UESTR	RW	S	132 chars.
ACCESS USER EXIT CODE	&UEXERR	RW	N	[-2..3]
DETERMINE IF UG/MANAGER IS RUNNING	&UGMGR	RO	N	[1,2]
CURRENT UGII VERSION	&UGVERS	RO	S	GE 3
UNIT OF MEASUREMENT Inches Millimeters	&UNIT	RO	N	[1,2] =1 =2
USER IDENTIFICATION STRING	&USERID	RO	S	30 chars.
VERTICAL TEXT JUSTIFICATION First line text alignment Mid line text alignment Bottom line text alignment	&VTXJST &TOP &MIDDLE &BOTTOM	RW C C C	N N N N	[1..3] =1 =2 =3
VIEW OF CURSOR FOR SELECTION Any view (smallest viewport) Work view	&VWCURS &ANY &WORK	RW C C	N N N	[1,2] =1 =2
RETURN VIEW OF LAST SELECTION	&VWSEL	RO	S	30 chars.
DISPLAY VIEW BORDERS AND NAMES Display Do not display	&WBORD &YES &NO	RW C C	N N N	[1,2] =1 =2
WORK COORDINATE SYSTEM OBJECT	&WCS	RW	E	



GPA Symbol Format Summary

Function	Symbol	Access Type	Data Type	Range
DISPLAY TEMPORARY WCS Display WCS Do Not Display WCS	&WCSDRW &YES &NO	RW C C	N N N	[1,2] =1 =2
LINE WIDTH DISPLAY Displayed Not displayed	&WIDDSP &YES &NO	RW C C	N N N	[1,2] =1 =2
CURRENT WORK LAYER	&WLAYER	RW	N	[1..256]
CURRENT WORK VIEW	&WVIEW	RO	N	GE 0
CHANGE WORK VIEW	&WORKVW	RW	S	30 chars.
EXTENSION LINE DISPLAY Both Lines First Line Only Second Line Only No Ext. Lines	&XLINE &BOTH &FIRST &SECOND &NONE	RW C C C C	N N N N N	[1..4] =1 =2 =3 =4



EDA Symbol Format Summary

Appendix F



All EDA symbols begin with the ampersand character (&) which is known as the prefix character.

Each EDA symbol is associated with either a unique object parameter or a constant and has an ACCESS TYPE, DATA TYPE, and RANGE.

ACCESS TYPE defines the READ/WRITE status of the EDA. EDAs contain data which can be extracted (READ) for use in your program. Some symbols allow you to directly alter the data (WRITE) by assigning the properly valued data to the EDA.

DATA TYPE defines whether a particular EDA is *object*, *number*, or *string* valued. The data associated with an EDA is object valued, it must be treated like an object as it is assigned or extracted. If you READ an object valued EDA and assign the data to a variable, the variable must be declared as an ENTITY.

Number valued EDAs, when assigned to a variable, do not need to be declared. In the following program, the system assumes that NUM is a numerical variable without it being declared as such.

String valued EDAs, when assigned to a variable, must be declared as a STRING.

RANGE defines the value which can be assigned to an EDA. Some EDAs have multiple numerical values which lie within their RANGE. These parameters changed (e.g. the value for the ellipse SUBTYP is always 1). The RANGE of string valued EDAs is defined in a number of characters allowed in the string (e.g. &ATTTL has a range of 10 characters).

The following are explanations of letters and/or characters which will be found in the *RANGE* column.

LT	--	Less Than
LE	--	Less Than Or Equal To
GT	--	Greater Than
GE	--	Greater Than Or Equal To
**	--	Any valid Unigraphics numerical value

The following are explanations of letters and/or characters which will be found in the *ACCESS* and *DATA TYPE* columns.

- N -- Number Valued
- S -- String Valued
- O -- Object (Entity) Valued
- RW -- Read/Write
- RO -- Read Only
- C -- Constant (value which may not be changed)

The EDAs are broken down into two categories: object dependent EDAs (entities are listed alphabetically with several EDAs for each), and all remaining EDAs (listed alphabetically by symbol name).

Function	Symbol	Access Type	Data Type	Range
ALL AROUND SYM DISP Displayed Not displayed	&ALLARN(<u>obj</u>) Yes No	RW	N	[1,2] =1 =2
ARC LENGTH SYMBOL Displayed Not displayed	&ALNSYM(<u>obj</u>) Yes No	RW	N	[1,2] =1 =2
ARROWHEAD DISPLAY DATA Arrowhead 1 type Arrowhead 2 type Filled arrowhead display Filled distance Included angle Dot diameter	&ARWDAT(<u>obj</u> , <u>n</u>)	RW	N	[1..6] =1 =2 =3 =4 =5 =6
NUMBER OF ASSOCIATED ENTITIES	&ASCCNT(<u>obj</u>)	RO	N	**
NAME DISPLAY LOCATION (X,Y,Z)	&ATDISL(<u>obj</u>)	RW	N(3)	± infinity
ALL ENTITY ATTRIBUTES	&ATTALL (<u>{obj PART 'name'}</u>)	RO	N	GTE 0
ATTRIBUTE TITLE	&ATTTL (<u>{obj PART 'name'}</u> , seqno[,data_type])	RO	S	50 char.
ATTRIBUTE VALUE	&ATTVL (<u>{obj PART 'name'}</u> , 'title' [,IFERR,label:] [,data_type])	RW	S	132 char.



Function	Symbol	Access Type	Data Type	Range
BLANKING STATUS Blanked Normal	&BLANK(obj)	RO	N	[1,2] =1 =2
B-SPLINE Vertex at U Weight at U Vertices Degree in U Direction	&BPOLE(obj,u) &BPOLEW(obj,u) &BSDATA(obj) &BSDATA(obj,DEGREE)	RW RW RO RW	N(3) N N N	 GT 0 [1..100] [1..100]
B-SURFACE Vertex at U,V Weight at U,V Vertices in U Direction Vertices in V Direction Degree in U Direction Degree in V Direction	&BPOLE(obj,u,v) &BPOLEW(obj,u,v) &BSDATA(obj) &BSDATA(obj,ROW) &BSDATA(obj,DEGREE) &BSDATA(obj,DEGREE,ROW)	RW RW RO RO RW RW	N(3) N N N N N	** GT 0 [1..100] [1..100] [1..24] [1..24]
COMPONENTS AND REF. SETS # of comp. members Name of nth member Comp. of which object is member Current date Current time Component part name Reference set name Component scale	 &CCOUNT(comp) &CMEM(comp,n) &MEMCMP(obj,IFERR,label:) &CDATE(comp) &CTIME(comp) &CPNAM(comp) &CRSNAM(comp) &CSCALE(comp)	 RO RO RO RO RO RW RW RO	 N O O S S S S N	 GE 0 9 char. 5 char. 132char 30 char GT 0
ARC Center (X,Y,Z) Radius Start angle End angle Start point (X,Y,Z) End point (X,Y,Z) Arc length X axis matrix values Y axis matrix values Z axis matrix values	&CENTER(obj) &RADIUS(obj) &SANG(obj) &EANG(ent) &SPOINT(obj) &EPOINT(obj) &LENGTH(obj) &XAXIS(obj) &YAXIS(obj) &ZAXIS(obj)	RW RW RW RW RO RO RO RO RO RO RO	N(3) N N N N(3) N(3) N N(3) N(3) N(3) N(3)	** ** ** ** ** ** ** ** ** ** **



Function	Symbol	Access Type	Data Type	Range
ELLIPSE Center (X,Y,Z)	&CENTER(obj)	RW	N(3)	**
Semimajor	&MAJOR(obj)	RW	N	**
Semiminor	&MINOR(obj)	RW	N	**
Tilt angle	&TANG(obj)	RW	N	**
Start angle	&SANG(obj)	RW	N	**
End angle	&EANG(obj)	RW	N	**
Start point (X,Y,Z)	&SPOINT(obj)	RO	N(3)	**
End point (X,Y,Z)	&EPOINT(obj)	RO	N(3)	**
X axis matrix values	&XAXIS(obj)	RO	N(3)	**
Y axis matrix values	&YAXIS(obj)	RO	N(3)	**
Z axis matrix values	&ZAXIS(obj)	RO	N(3)	**
HYPERBOLA				
Center (X,Y,Z)	&CENTER(obj)	RW	N(3)	**
Half traverse	&HTRAV(obj)	RW	N	**
Half conjugate	&HCONJ(obj)	RW	N	**
Tilt angle	&TANG(obj)	RW	N	**
Minimum Y dist.	&YMIN(obj)	RW	N	**
Maximum Y dist.	&YMAX(obj)	RW	N	**
Start point (X,Y,Z)	&SPOINT(obj)	RO	N(3)	**
End point (X,Y,Z)	&EPOINT(obj)	RO	N(3)	**
X axis matrix values	&XAXIS(obj)	RO	N(3)	**
Y axis matrix values	&YAXIS(obj)	RO	N(3)	**
Z axis matrix values	&ZAXIS(obj)	RO	N(3)	**
CHARACTER PARAMETERS	&CHRDAT(obj,n)	RW	N	[1..4]
Character size				=1
Aspect ratio				=2
Space factor				=3
Line factor				=4
NATIVE LANGUAGE MENU CHARACTER SETS	&CHRSET(IP1, IP2)	RO	N	[0..4]
COLOR NAME	&CLRNME(num)	RO	S	10 char.
COLOR NUMBER	&CLRNUM(name)	RO	N	[1..15]

Function	Symbol	Access Type	Data Type	Range
COLOR	&COLOR(<u>obj</u>)	RW	N	[1..15]
Blue	&BLUE	C	N	=1
Green	&GREEN	C	N	=2
Cyan	&CYAN	C	N	=3
Red	&RED	C	N	=4
Magenta	&MAGENT	C	N	=5
Yellow	&YELLOW	C	N	=6
White	&WHITE	C	N	=7
Olive	&OLIVE	C	N	=8
Pink	&PINK	C	N	=9
Brown	&BROWN	C	N	=10
Orange	&ORANGE	C	N	=11
Purple	&PURPLE	C	N	=12
Dark red	&DKRED	C	N	=13
Aquamarine	&AQUAMR	C	N	=14
Gray	&GRAY	C	N	=15
LINE DENSITY	&DENS(<u>obj</u>)	RW	N	[1..3]
Normal				=1
Thick (Heavy)				=2
Thin				=3
TEXT	&DMTEXT(<u>obj</u>)	RO	S	132 char. Per Line
EDGE TYPE	&EDGTYP (<u>obj</u> [,IFERR,label:])	RO	N	[0,3,5, 6,9]
No curve attached				=0
Line				=3
Arc/Circular				=5
Conic (ellipse)				=6
Spline				=9
FIND THE NTH ENTITY WITH THE GIVEN NAME	&ENAME(n,'name' [,IFERR,label:])	RO	O	
RUNOUT FILLED ARROWHD	&FILRUN(<u>obj</u>)	RW	N	[1,2]
Displayed	Yes			=1
Not displayed	No			=2
FONT NAME	&FNTNME	RO	S	16 char.
FONT NUMBER	&FNTNUM	RO	N	[1..14]





Function	Symbol	Access Type	Data Type	Range
FONT Solid line Dashed Phantom Centerline Dotted Long Dashed Dotted Dashed	&FONT(<u>obj</u>) &SOLID &DASHED &PHANTM &CLINE	RW C C C C	N N N N N N N	[1..4] =1 =2 =3 =4 =5 =6 =7
FONT NAME STRING	&FONTN(<u>obj</u>)	RW	S	30 char.
GROUP Number of entities Member object at n Group of object	&GCOUNT(<u>obj</u>) &GENT(<u>obj</u> ,n) &GROUP(<u>obj</u>)	RO RO RO	N O O	**
GROUPING STATUS Grouped Ungrouped	&GRSTAT(<u>obj</u>)	RO	N	[1,2] =1 =2
LAYER	&LAYER(<u>obj</u>)	RW	N	[1..257]
LINE BETWEEN ARROWS Display Do not display	&LNARRW(<u>obj</u>) &YES &NO	RW C C	N N N	[1,2] =1 =2
LINE WIDTH Normal Thick (Heavy) Thin	&LWIDTH (<u>obj</u> [,IFERR,label:])	RW	N	[1..3] =1 =2 =3
GLOBAL LAYER SELECTABILITY MASK Layer is selectable Layer is not selectable	&LYRSEL(layer number [,IFERR,label:]) &YES &NO	RW	N C C	[1,2] =1 =2
GLOBAL LAYER VISIBILITY MASK Layer is visible Layer is not visible	&LYRVIS(layer number [,IFERR,label:]) &YES &NO	RW	N C C	[1,2] =1 =2
LAYER VISIBILITY IN A VIEW Layer is visible in view Layer is not visible	&LYRVVW('view name', layer number [,IFERR,label:]) &YES &NO	RW	N C C	[1,2] =1 =2
ENTITY NAME	&NAME({ <u>obj</u> 'name'})	RW	S	30 char.

Function	Symbol	Access Type	Data Type	Range
NUMBER OF ENTITY ATTRIBUTES	&NATTR (<u>obj</u> PART 'name'}, [data_type])	RO	N	GT 0
DRAFTING ENTITY ORIGIN	&ORIGIN(<u>obj</u>)	RW	N(3)	± Infinity
PATTERNS: Origin and orientation origin (X,Y,Z)	&ORIGIN(<u>obj</u>)	RO	N(3)	**
X axis matrix values	&XAXIS(<u>obj</u>)	RO	N(3)	**
Y axis matrix values	&YAXIS(<u>obj</u>)	RO	N(3)	**
Z axis matrix values	&ZAXIS(<u>obj</u>)	RO	N(3)	**
COORDINATE SYSTEM origin (X,Y,Z)	&ORIGIN(<u>obj</u>)	RO	N(3)	**
X axis matrix values	&XAXIS(<u>obj</u>)	RO	N(3)	**
Y axis matrix values	&YAXIS(<u>obj</u>)	RO	N(3)	**
Z axis matrix values	&ZAXIS(<u>obj</u>)	RO	N(3)	**
OFFSET SURFACE Offset Distance	&OSDIST(<u>obj</u>)	RW	N	**
PATTERNS: Max/Min Box Display	&PATBOX(<u>ptrn</u> [,IFERR,label:])	RW	N	[1,2]
Status Display	&YES	C	N	=1
Do Not Display	&NO	C	N	=2
PATTERNS: Control Point Display	&PATCPT(<u>ptrn</u> [,IFERR,label:])	RW	N	[1,2]
Status Display	&YES	C	N	=1
Do Not Display	&NO	C	N	=2
PATTERNS: Data Retrieval	&PATDAT (<u>obj</u> [,IFERR,label:])	RO	N	[1,2]
Successful	&YES	C	N	=1
Not Successful	&NO	C	N	=2
PATTERNS: Origin Marker Display	&PATMRK (<u>ptrn</u> [,IFERR,label:])	RW	N	[1,2]
Status Display	&YES	C	N	=1
Do Not Display	&NO	C	N	=2
PATTERNS: Return Part Name	&PATPRT (<u>obj</u> [,IFERR,label:])	RO	S	30 char.
PATTERNS: Return Scale	&PATSCAL (<u>obj</u> [,IFERR,label:])	RO	N	GT 0
PLANE Vertex (X,Y,Z)	&POINT(<u>obj</u>)	RW	N(3)	**
Normal vector (I,J,K)	&NORMAL(<u>obj</u>)	RO	N(3)	**





Function	Symbol	Access Type	Data Type	Range
POINT Coordinates (X,Y,Z)	&POINT(<u>obj</u>)	RW	N(3)	**
FIND THE PROTOTYPE OF THE INPUT OBJECT	&PROTO(<u>obj</u>)	RO	O	
SOLID Density	&SDENS(<u>obj</u>)	RW	N	GE 0
SCRATCH FILE NAME	&SFNAME (file#[,IFERR,label:])	RW	S	132 char.
NUMBER OF EDGES OR FACES	&SOLDAT (ent,{FACE EDGE} [,IFERR,label:])	RO	N	GE 0
LINE Start point End point Length	&SPOINT(<u>obj</u>) &EPOINT(<u>obj</u>) &LENGTH(<u>obj</u>)	RW RW RO	N(3) N(3) N	** ** **
READ STATE OF AN ANNOTATION	&RETFLG	RO	N	[1,2]
SPLINE Start point (X,Y,Z) End point (X,Y,Z) Number of points Coord's of point n Tan. vect. at point n Chord sum at point n Spline Type General Closed	&SPOINT(<u>obj</u>) &EPOINT(<u>obj</u>) &NUMSPT(<u>obj</u>) &POINT(<u>obj</u> ,n) &TVECT(<u>obj</u> ,n) &PAR(<u>obj</u> ,n) &SUBTYP(<u>obj</u>)	RO RO RO RO RO RO RO	N(3) N(3) N N(3) N(3) N N	** ** ** ** ** ** ** [1,2] =1 =2
SUBTYPE OF CONIC Ellipse Hyperbola Parabola	&SUBTYP(<u>obj</u>)	RO	N	[1...3] =1 =2 =3
SUBTYPE OF SOLID ENT. Solid body Face Edge	&SUBTYP(<u>obj</u>)	RO	N	[0,2,3] =0 =2 =3

Function	Symbol	Access Type	Data Type	Range
SUBTYPE OF DIMENSIONS Horizontal Vertical Parallel Cylindrical Perpendicular Angular, minor angle Angular, major angle Arc length Radius Diameter Hole Concentric circles Ordinate, horizontal Ordinate, vertical Assorted parts	&SUBTYP(<u>obj</u>)	RO	N	[1..15] =1 =2 =3 =4 =5 =6 =7 =8 =9 =10 =11 =12 =13 =14 =15
SUBTYPE OF DRAFT AIDS Note Label ID symbol FMPOS symbol Center line Crosshatching Assorted parts	&SUBTYP(<u>obj</u>)	RO	N	[1..7] =1 =2 =3 =4 =5 =6 =7
TRAILING ZERO SUPPRESSION Displayed Not displayed	&SUPRES(<u>obj</u>) Yes No	RW	N	[1,2] =1 =2
DETERMINE IF AN OBJECT IS A SILHOUETTE	&ISSIL(<u>object</u> [, IFERR, label:])	RO	N	[0,201]
DETERMINE IF AN OBJECT IS A SECTION EDGE	&ISSXED(<u>object</u> [, IFERR, label:])	RO	N	[0,199]
DETERMINE IF AN OBJECT IS A SECTION LINE	&ISSXL(<u>object</u> [, IFERR, label:])	RO	N	[0,202]
SYMBOL PARAMETERS	&SYMDAT(<u>obj</u> , <u>seqnum</u>)	RW	N	Any Real
SYMBOL NAME IN DRAFTING ENTITY	&SYMDFT(<u>obj</u> , <u>index</u>)	RW	S	8 char.
DISTANCE BETWEEN TEXT AND DIAMETER SYMBOL	&SYMDIS(<u>obj</u>)	RW	N	GE 0
NUMBER OF SYMBOLS IN A DRAFTING ENTITY	&SYMNUM(<u>obj</u>)	RO	N	**





Function	Symbol	Access Type	Data Type	Range
TOLERANCE – ENGLISH	&TOL(<u>obj</u> , <u>n</u>)	RW	N	± Infinity
TOLERANCE – SI	&TOLM(<u>obj</u> , <u>n</u>)	RW	N	± Infinity
QUERY/MODIFY COLOR SETTINGS	&TYPCLR(TYPE,number [,SUBTYP,number] [,PROP,property] [,IFERR,label:])	RW	N	[-1..15]
QUERY/MODIFY LINE FONT SETTINGS	&TYPWID(TYPE,number [,SUBTYP,number] [,PROP,property] [,IFERR,label:])	RW	N	[-1..7]
QUERY/MODIFY LINE WIDTH SETTINGS	&TYPFNT(TYPE,number [,SUBTYP,number] [,PROP,property] [,IFERR,label:])	RW	N	[-1..3]
ENTITY TYPE	&TYPE(<u>obj</u>)	RO	N	2-202
PARABOLA Vertex (X,Y,Z) Focal length Tilt angle Minimum Y dist. Maximum Y dist. Start point (X,Y,Z) End point (X,Y,Z) X axis matrix values Y axis matrix values Z axis matrix values	&VERTEX(<u>obj</u>) &FOCAL(<u>obj</u>) &TANG(<u>obj</u>) &YMIN(<u>obj</u>) &YMAX(<u>obj</u>) &SPOINT(<u>obj</u>) &EPOINT(<u>obj</u>) &XAXIS(<u>obj</u>) &YAXIS(<u>obj</u>) &ZAXIS(<u>obj</u>)	RW RW RW RW RW RO RO RO RO RO	N(3) N N N N N(3) N(3) N(3) N(3) N(3)	** ** ** ** ** ** ** ** ** **
VERTICAL TEXT JUSTIFICATION Top Middle Bottom	&VTXJT(<u>obj</u>)	RW	N	[1..3] =1 =2 =3
VIEW DEPENDENT STATUS OF AN ENTITY	&VWDEP(<u>obj</u> [,IFERR,label:])	RW	S	30 char.

Word/Symbol List

Appendix G

Major Word List

ABSF	CPARF	DOTF	FILLET	JUMP	PATRET
ACOSF	CPATT	DRAW	FILSRF	LABEL	PATUPD
ADDPL	CPOSF	DRAWC	FMOVE	LAYC	PATLER
ADIM	CPSET	DRAWD	FMPOS	LAYD	PLANE
ANGLF	CRDIR	DRAWE	FNDSTR	LAYE	PHMOD
ANLSIS	CREATE	DRAWN	FONT*	LAYER*	PHREAD
APPEND	CROSSF	DRAWV	FONDF	LAYN	PLCSYM
ARCDIM	CRRFST	DVIEW	FPRINT	LAYR	PLMODE
ASATT	CRSEWV	EDGVER	FRACT	LCASE	PLNOTE
ASCENT	CRVTRM	EDIT	FRDIM	LDEL	PLOT
ASCII	CRTWRT	EDITPL	FSTR	LDIM	PLOUT
ASGNFT	CSIZE	EDRFST	FSTRL	LENF	POINT*
ASINF	CSYS	EDTXHT	FTERM	LINE	POS
ATANF	CTANF	EJECT	GCONIC	LOGF	PRINT
AUTOSF	CTRIM	ELLIPS	GENDIM	LSTDEV	PROC
BASURF	CYLDIM	ELSE	GETL	MAP	PROJ
BATCH	CYLNDR	ELSEIF	GPOS	MATRIX	RCOMP
BCURVE	DATE	ENCONT	GRDDSP	MAXF	RDIM
BLANK*	DCLOSE	ENDIF	GRIPSW	MCHOOS	READ
BLEND	DDIM	ENUM	GROUP	MESH	REFCNT
BLENFX	DEACT	EPROP	GRPEDT	MESSG	REFMEM
BLSTR	DECPL	EXPF	HALT	MINF	REGF
BPLANE	DELETE	EXPCRE	HATCH	MODF	RELDST
BOUND	DELIM	EXPDEL	HDIM	NEWCTE	REMVPL
BSURF	DELNAM	EXPEDT	HYPERB	NEXTE	RENAME
CALL	DENS*	EXPEXP	IDENT	NEXTN	RENORM
CAT	DEPTH	EXPEXP	IDSYM	NODE	REPSTR
CCDIM	DEVCHK	EXPIMP	IF	NOTE	RESEQ
CHAIN	DFSTR	EXPLIS	IFTHEN	OBTAIN	RESET
CHIDC	DFTSYM	EXPRNM	INEXTE	ODIM	RETURN
CHKSOL	DIAG	FACMOV	INEXTN	OFFCRV	REVSFR
CHOOSE	DIMBP	FAPEND	INHAT	OFFSRF	RLDSRF
CHRSTR	DIMCON	FCOMP	INTERS	OODIM	RPAIN
CIRCLE	DIMPAR	FCOPY	INTF	OOMGN	RPATT
CLINE	DISTF	FDEL	INTFER	PARABO	RPATTG
CNEXT	DLATT	FETCH	INTSEC	PARABO	RTVBLNK
CMPSTR	DNEXT	FHMOD	ISOCR	PARAM	SBFONT
CONE	DO	FHREAD	ISTR	PARTOP	SCALVF
COSF	DOPEN	FILE	ISTR	PARTST	SCLOSE
				PATEXP	SECT



SDDUF	SOLCYL	SPARF	SXSEGD	UNBLNK	VIEWC
SDDVF	SOLEDG	SPHERE	TABCYL	UNGRP	VIEWD
SEW	SOLENT	SPLINE	TEXT	UNITE	VIEWE
SILHO	SOLEXT	SPLIT	TIME	UNITF	VIEWLC
SIMCRV	SOLPIX	SPOSF	TPMOD	UPDATE	VIEWN
SINF	SOLREV	SQRTF	TRANSF	UPGRAD	VIEWV
SNORF	SOLSPH	SSURF	TYPF	USED	VLENF
SOLBLK	SOLTOR	STORE	UDFSYM	VALF	WRITE
SOLBOX	SOLTUB	SUBSTR	UFARGS	VDCPRM	XSPAWN
SOLCON	SOPEN	SUBTRA	UGMGRD	VDEDIT	XTERM
SOLCUT	SORT	SXLDAT	UGMGRE	VIEW	

* These words are also used as *ENTITY DATA ACCESS* symbols.

Minor Word List

ACCRCY	BOTH	DASH	EDGE	HEX	LIGHT
ACTIVE	CANCEL	DATUM	ELAYER	HEIGHT	LINK
ADD	CAREA	DCTRC	ELOG	HIGH	LINNO
AFILL	CAT	DCUE	END	HOLE	LNODAL
AFTER	CENTER	DECEND	ENDOF	HORIZ	LPT
ALL	CGM	DECIM	ENDPTH	HORVER	LTHERM
ALTACT	CHAMFR	DECLRV	ENTER	IFEND	LOW
ALWAYS	CHC	DEFLT	EPARAM	IFERR	LWIDTH
AMBI	CIR	DEGREE	ERASE	IGNORE	MAG
ANCHOR	CHECK	DELCH	EXPAR	IN	MAJOR
ANGLE	CLBLK	DELETE	EXT	INACT	MATERL
ANTALI	CLDLT	DELTA	EYEPT	INCHES	MAX
APEX	CLOSED	DENS	FACE	INCR	MCHFMT
APPEND	CLSF	DESCR	FBOLT	INQUIR	MEDIA
APPROX	CLUBLK	DHEX	FCIRC	INT	MEMBER
ARCLEN	CMETER	DIFF	FILLET*	INTENS	MESH
ARROW	CNT	DISK	FINISH	INTOF	METER
ARROWS	COLOR	DIST	FIRST	INVERS	MIN
ASCEND	COLUMN	DIAMTR	FNAME	INVSBL	MIRROR
ATANGL	COMMON	DISPL	FOLLOW	IPM	MMETER
ATRBTS	COMP	DOGLEG	FONT	LAG	MMPM
AUTO	COMPIL	DOT	FONTN	LINEAR	MODEL
AUTOZ	CONCUR	DRILL	FREE	KGM	MOVE
AXIS	CONSRF	DSAVE	FROM	KNOT	MTAPE
BACK	COPIES	DSCALE	FRONTZ	LARGE	MULTI
BACKZ	CROSS	DSQR	FROZEN	LAYER*	NEWEST
BEFORE	CRT	DSTCTR	GCM	LBFT	NEXTL
BLANK	CSYS	DSTNRM	GENCRV	LBIN	NONE
BLEND	CURSOR	DSURF	GENSRF	LEAD	NORM
BYLAYR	CURVE	DVSTAT	GEOM	LEADER	NOTRIM
BND	CUSTDF	EARCL	GROOVE	LEFT	NOVIEW
BNDRY	CWRK	EDGCRV	GROUP*	LELEM	NRML
					NULL

OBLIQ	PLMODS	REPL	SLOG	THIN	VERIFY
OBLNG	PLOT	REPLAC	SOLID	THREAD	VERT
OFF	PLTTOL	REPORT	SPEC	THRU	VIEW*
OFFCPT	POLAR	REMOVE	SQR	TILT	VMODS
OFFCYL	POWR	RESL	SPINE	TNGNT	VOLBND
OFFDST	PRIMA	REST	SRFDSP	TO	VOLREV
OFFPT	PRINT	RESULT	START	TOLER	VSBL
OMIT	PROFIL	RETCAM	STATUS	TOP	WITH
ON	PROP	RETRVE	STEP	TOTAL	WORK
OPEN	PROSOL	RHO	STR	TRACRV	XAXIS
ORDER	PRTFIL	RIGHT	STRING	TRANSL	XCAXIS
ORIENT	PRTNUM	ROOT	SUBDIV	TRIANG	XHATCH
ORIG	PRTREV	ROUGH	SUMM	TRIDWN	XLARGE
ORIGIN	PRTTYP	ROW	SURFC	TRIM	XSMALL
OS	PSTAN	RTD	SWEEP	TRIUP	XYPLAN
OUT	PSURF	RUN	SWNEGU	TXT	XYROT
OVERW	PSTAN	RWRK	SWNEGV	TXTANG	XZPLAN
PARLEL	PSURF	SAME	SWPOSU	TYPE	YAXIS
PART	PTTOPT	SCALE	SWPOSV	UDIR	YCAXIS
PAST	QTY	SCOPE	SWPSRF	UFUN	YLARGE
PATPNT	QUAD	SECOND	SWRK	UGDFLT	YSMALL
PAUSE	QUICK	SHELL	SYMANG	UPDATE*	YZPLAN
PBOLT	RABS	SIDE	SYMBOL	USERID	YZROT
PCIRC	RADIUS	SILCRV	SYMMET	USING	ZIG
PERP	RATIO	SILHO	TANTO	UTIL	ZIGZAG
PERPTO	RECURS	SIZE	TAPE	VDEP	ZLARGE
PERSP	REF	SKIPVL	TEMP	VDIR	ZSMALL
PHANTM	RELNO	SLAYER	THICK	VECT	ZXROT

* These *MINOR WORDS* are also used as *MAJOR WORDS*.

GPA Symbol List

&AADIST	&ANGFLG	&ARROW2	&BGCLR	&CENTER	&COMBN
&AAPL	&ANGTOL	&ASIZE	&BILD	&CFONT	&COMMA
&ABOVE	&APSITE	&ASPECT	&BLDIS	&CHAIN	&CON
&ACTIVE	&AQUAMR	&ATTDIS	&BLNKPT	&CHTOL	&CONAME
&ACTPRT	&ARCCTR	&ATTYPE	&BLUE	&CLINE	&CONCRT
&ADJST	&ARCHCR	&AUNIT	&BOTC	&CLOSED	&CONFRM
&ADV	&ARCHFI	&AUTO	&BOTH	&CLPTOL	&CONINC
&AFTER	&ARDSP	&BASIC	&BOTL	&CLRPL	&CONT
&AGRAIN	&ARFLDT	&BCDIS	&BOTR	&CLRST	&CONTIN
&ALARND	&ARIANG	&BCFLG	&BRASS	&CLW	&COOLNT
&ALL	&ARLNSM	&BCHIP	&BROWN	&CMPFRM	&CORE
&ALLBUT	&ARRIN	&BCONT	&CAREA	&CNMODE	&CORK
&ALUM	&ARROUT	&BEFORE	&CCLW	&CNODE	&CRDATE
&ANGDIS	&ARROW	&BELOW		&CNROLL	&CROSS



&CRT	&DSANG	&FNAME	&INDIC	&MIDR	&PERINC
&CRTIME	&DSTATE	&FOLLOW	&INTOL	&MINSTP	&PERIOD
&CSIZE	&DSTXR2	&FONT	&INVERS	&MINTCK	&PHANTM
&CSLANT	&DSVEC	&FONTN	&IPM	&MM	&PI
&CURDIR	&DTDECP	&FORMAT	&IPR	&MMPM	&PINK
&CURDRW	&DTOLC	&FORWRD	&IRON	&MMPR	&PITCH
&CURLAY	&DTOLS	&FRAC	&JUNCT	&MODEL	&PITINC
&CUTANG	&DUAL	&FREE	&LADATE	&MODULE	&PITYPE
&CUTCOL	&DUNIT	&FRONT	&LANG	&MRELAB	&PLMACH
&CUTFED	&DUNITD	&FSTAT	&LATIME	&MSVEC	&PLTFRM
&CUTTYP	&DUPCAM	&FSTOCK	&LCHECK	&MTPATT	&PNAME
&CUTUN	&EARTH	&FSYS	&LCTPER	&MXRPM	&PNTARR
&CWCS	&ENDPT	&FTYPE	&LCTPTS	&MVDISP	&PNTDS
&CWCSL	&ENGANG	&FTYPED	&LDEV	&NCHASE	&PNTFED
&CWCSR	&ENGCOL	&FULL	&LDINC	&NO	&PNTLN
&CYAN	&ENGDIS	&GAPFAC	&LEAD	&NODUAL	&PNTSP
&DASHED	&ENGFED	&GBNDSP	&LEADER	&NOFOL	&PNTTF
&DASHSZ	&ENGFI	&GBNDSR	&LEFT	&NONE	&PNTTL
&DDDECP	&ENGPL	&GBSTR	&LENGTH	&NORMAL	&POINTS
&DDECPL	&ENGTYP	&GENDPT	&LENGTP	&NORMPS	&PROFIL
&DEC	&ENGVEC	&GLRMV	&LENGVC	&NOTOL	&PROVEC
&DECIM	&ENSITE	&GLRPT	&LIMDA	&NPASS	&PSANG
&DECPL	&ENTCLR	&GLRST	&LIMDB	&NSTART	&PSPEC
&DECPT	&ENTDIA	&GOUGE	&LIMSA	&NULENT	&PSURF
&DEG	&EQUAL	&GRAY	&LIMSB	&NULSTR	&PSVEC
&DEGM	&ERRMOD	&GREEN	&LINFAC	&NUMOPS	&PTYPE
&DEGMS	&ERROR	&GROOVE	&LIQUID	&NVP	&PURPLE
&DELREV	&EXCLIB	&GSIZE	&LMDATE	&OFF	&PVAR
&DELSEC	&EXPAND	&GSIZEA	&LMTIME	&OFFSET	&RAD
&DENOM	&FACE	&GSIZER	&LNARRW	&OLIVE	&RADSYM
&DENOMD	&FASTF	&GSIZEX	&LNUM	&OMIT	&RANGE
&DEPTH	&FCTCOL	&GSIZEY	&LOCRMV	&ON	&RANGTX
&DESCR	&FCTDEP	&GSNAP	&LOCRPT	&OPEN	&RAPCOL
&DIASYM	&FCTFED	>YPE	&LOCRST	&ORANGE	&RAPFED
&DIR	&FCTPER	&GUTTYP	&LOW	&OSDIST	&REAR
&DIRECT	&FCTPTS	&HANGLE	&LOWPL	&OTHER	&RAPUN
&DISTDT	&FDUNT	&HDIST	&LPT	&OUTOL	&RED
&DISTOL	&FEDRAT	&HEAVY	&LRETVC	&OWNER	&REF
&DISTXD	&FEDUN	&HIGH	&LRTTYP	&OWNER	&REFRAC
&DISTXR	&FILARW	&HMAT	&LWIDTH	&PARATT	®
&DKRED	&FINDIR	&HORVER	&M	&PATBOX	®A
&DLWID	&FINISH	&HSEL	&MACH	&PATCPT	®B
&DOLLIB	&FINPAS	&HSELTX	&MAGENT	&PATERR	®C
&DOT	&FIRST	&HTOLER	&MARBLE	&PATFIL	®D
&DOTDIA	&FIXED	&IDSIZE	&MAXSTP	&PATMRK	®E
&DPREL	&FLARDP	&INACT	&MCHFMT	&PATSHD	®F
&DRILL	&FLEN	&INC	&MED	&PCLASS	®G
&DRSITE	&FMODE	&INCH	&MIDC	&PDDIS	®H
&DRTOOL	&FMSIZE	&INCTYP	&MIDL	&PDRILL	®I
				&PER	®J

®K	&SDENS	&STKX	&TEMP	&TQFRAC	&VWCURS
®L	&SDUNIT	&STKY	&TENDL	&TXALIN	&VWSEL
&RELNO	&SECOND	&STPCOL	&TERM	&TXANGL	&WBORD
&REPEAT	&SFM	&STPFED	&TEXBOX	&TXARR	&WCS
&RETANG	&SGRIDU	&STPPER	&TEXT	&TXHOR	&WCSDRW
&RETCOL	&SGRIDV	&STPPTS	&TEXTOR	&TXJUST	&WGRAIN
&RETDIS	&SIDE	&STPTOL	&THDANG	&TXOVER	&WHITE
&RETFED	&SIMPLE	&STEPTY	&THIN	&TXTANG	&WLAYER
&RETPL	&SINTOL	&STUBB	&THREAD	&UENUM	&WORKVW
&RETTYP	&SMETH	&SUBDEP	&TLWID	&UESTR	&WVIEW
&RETVEC	&SNDINS	&SUBMOD	&TNAME	&UESTR	&XCLRPL
&RGNCN	&SOLID	&SUPRES	&TNODE	&UGVERS	&XCLRST
&RIGHT	&SOLIDF	&SURFC	&TOLER	&UNDEF	&XDIRCT
&ROCK	&SOUTOL	&SWNEGU	&TOLOW	&UNILA	&XLINE
&ROUGH	&SPCSZ	&SWNEGV	&TOLOWM	&UNILB	&XY
&RNFILL	&SPDIR	&SWPOSU	&TOLS	&UNIT	&YCLRPL
&RNDIST	&SPDIRL	&SWPOSV	&TOLUP	&UNSTP	&YCLRST
&RPM	&SPKTOL	&SYMANG	&TOLUPM	&USERID	&YDIRCT
&RTD	&SPSPD	&SYMBSZ	&TOOL	&VAR	&YELLOW
&RUBBER	&SSTOCK	&SYMDIS	&TOPC	&VARINC	&YES
&SADDLE	&STD	&SYMSCA	&TOPL	&VECPL	&YX
&SAFE	&STEEL	&SYMTXT	&TOPR	&VECTOR	&ZIG
&SAND	&STEEP	&SYSCLR	&TOSITE	&VIEW	&ZIGZAG
&SBDISP	&STEP	&TAXL	&TPANLY	&VP	&ZZ
&SCALLP	&STEPDR	&TAXTYP	&TPU	&VTXJST	&ZZLIFT
&SCHEMA	&STKTOT	&TDECPL	&TPUINC		

EDA Symbol List

&ALANSYM	&CLRNUM	&FNTNUM	&MEMCPC	&POINT*	&TLDFT
&ALLARN	&CMEM	&GCOUNT	&MINOR	&PROTO	&TLDIA
&AQUAMR	&COLOR	&GENT	&NAME	&PURPLE	&TLHGT
&ARWDAT	&CPNAM	&GRAY	&NATTR	&RADIUS	&TOL
&ASCNT	&CRSNAM	&GREEN	&NORMAL	&RED	&TOLM
&ATDISL	&CSCALE	&GROUP*	&NUMSPT	&SANG	&TVECT
&ATTALL	&CTIME	&GRSTAT	&OLIVE	&SDENS	&TYPE
&ATTTL	&CYAN	&HCONJ	&ORANGE	&SFNAME	&VERTEX
&ATTVL	&DENS*	&HTRAV	&ORIGIN	&SOLDAT	&VTXJT
&BLANK*	&DMTEXT	&LAYER*	&OSDIST	&SPOINT	&VWDEP
&BLUE	&EANG	&LENGTH	&PAR	&SUBTYP	&WHITE
&BROWN	&EDGTYP	&LNARRW	&PATBOX	&SUPRES	&XAXIS
&CCOUNT	&EPOINT	&LWIDTH	&PATCPT	&SYMDAT	&YAXIS
&CDATE	&FILRUN	&LYRSEL	&PATDAT	&SYMDFT	&YELLOW
&CENTER	&FOCAL	&LYRVIS	&PATMRK	&SYMDIS	&YMAX
&CHRDAT	&FONT*	&LYRVVW	&PATPRT	&SYMNUM	&YMIN
&CHRSET	&FONTN	&MAGENT	&PATSCL	&TANG	&ZAXIS
&CLRNME	&FNTNME	&MAJOR	&PINK	&TLCOR	

* These symbols are also *MAJOR WORDS*.





Index

A

ABSF, 12–5
 ACOSF, 12–8
 ADIM, 15–11
 Arithmetic Functions, 12–5
 ASATT, 16–9
 ASCII, 12–18
 ASCII Value of a Character, 12–18
 ASINF, 12–8
 Assign, object attributes, 16–9
 Assigning Values
 Entity Variables, 2–19
 Numeric Variables, 2–18
 String Variables, 2–17
 ATANF, 12–8
 ATDISL – EDA, name display location, 16–22
 Attribute title, 16–12
 Attribute value, 16–16
 Attributes, verify name, 16–5
 ATTTL – EDA, attribute title, 16–12
 ATTVL – EDA, attribute value, 16–16

B

Blank Characters, create, 12–13
 BLEND, 8–16
 BLOCKIF, Block If, 5–16
 BLSTR, 12–13

C

CALL, 13–3
 Call Statement, 13–3
 CAT, 10–5
 CAT, Create Layer Category, 10–5
 CATD, 10–8
 CATD, Delete Category, 10–8
 CATE, 10–7

CATE, Edit Category, 10–7
 Categories and Layer Control, 10–2
 CATV, Query Category, 10–9
 Choose Multiple Options, 7–15
 Choose Single Option, 7–11
 CHOOSE, Choose Single Option, 7–11
 CHRSTR, 12–19
 Circle, 4–2
 Center Coordinates Radius, 4–3
 Center Point
 Point on the Arc, 4–9
 Radius, 4–5
 Tangent to a Line, 4–7
 EDA, 4–19
 Through Three Points, 4–11
 CLINE, 15–27
 Compiler directives, detect undeclared variables,
 2–23
 Compiling, 1–5
 Compiling GRIP Programs, 1–4
 Convert Integer to Character String, 12–15
 Convert Real Number to Character String, 12–16
 Convert String to Real Number, 12–17
 COSF, 12–7
 CREATE, 9–4

D

Data Statements, 2–16
 Debugger, B–2
 Clear Breakpoints, B–6
 Continue, B–9
 Examine Variables/Entities, B–6
 List Options, B–9
 Program Abort, B–9
 Set Breakpoints, B–4
 Set Variable, B–8
 Single Step, B–4
 Single Step/Into, B–4
 Declarations, GRIPSW, 2–23
 Declaring Variables, 2–7
 DELETE, 10–23
 Delete
 entity attributes, 16–19



object name, 16–8
DELETE, Delete, 10–23
DELNAM, 16–8
Detect undeclared variables, 2–23
Dimensions, 15–2
Display Control, 10–19
DLATT, 16–19
DMTEXT, 15–30
DO, Program Loop, 5–9
Drafting Entity Origin, 15–29
Drafting Objects on Drawings, creating, 15–5
DRAW, Control Object Display, 10–20
DRAWC, Create a Drawing, 14–4
DRAWD, Delete a Drawing, 14–11
DRAWE, 14–6
Drawing
 Creation of, 14–4
 Deleting, 14–11
Drawings, member view names, specifying, 14–3
DRAWV, 14–7

E

EDA Symbol List, G–5
EDGTYP, 8–16
Edit, 1–4
ENAME – EDA, find the nth object with the given name, 16–6
ENSITE, 15–20
Enter Parameters, 6–15
Enter Text, 7–8
Entity attributes, delete, 16–19
Entity data access symbols (EDAs)
 &ATDISL, 16–22
 &ATTTL, 16–12
 &ATTVL, 16–16
 &CENTER, 4–19
 &COLOR, 4–21
 &DMTEXT, 15–30
 &EANG, 4–19
 &EDGTYP, 8–18
 &ENAME, 16–6
 &EPOINT, 3–22, 4–19
 &FONT, 4–21
 &LAYER, 4–22
 &LENGTH, 3–22, 4–19
 &LWIDTH, 4–22
 &NAME, 16–3
 &ORIGIN, 15–29
 &RADIUS, 4–19
 &SANG, 4–19
 &SDENS, 8–18
 &SOLDAT, 8–18
 &SPOINT, 3–22, 4–19
 &SUBTYP, 10–16
 &TYPE, 10–16
 &XAXIS, 4–19
 &YAXIS, 4–19
 &ZAXIS, 4–19
Entity Independent EDA's, 4–21
Entity Type, 10–15
Entity Variables, 2–12
ENUM, 16–5
Error Handling, 5–23
 Fatal, Method 1, 5–24
 Fatal, Method 2, 5–25
 Non-Fatal, 5–23
Errors, 1–8
EXPF, 12–5
Extract a Portion of a Character String, 12–22

F

FDEL, 9–8
FETCH, 9–3
FILE, 9–5
File access statements
 close file, 9–6
 create file, 9–4
 file, 9–5
 retrieve file, 9–3
File edit
 file pointer control: RESET, 9–18
 read text, 9–19
 write text, 9–21
File maintenance statements, delete file, 9–8
Fillet, 4–12
 Two Entities, Center Point, 4–13
 Two Entities, Positional Modifiers, 4–16
Find the nth entity with given name, 16–6
FNDSTR, 12–23
FTERM, 9–6

G

Global parameter access symbols (GPAs)

&ACTPRT, 10–28
 &DECPL, 10–28
 &SDENS, 8–18
 &SDUNIT, 8–18
 &UNIT, 10–30
 &USERID, 10–30
 &WLAYER, 10–29
 &WORKVW, 10–29

GPA constants

&ARRIN, 15–19
 &ARROUT, 15–19
 &AUTO, 15–19
 &BOTC, 15–21
 &BOTH, 15–18
 &BOTL, 15–21
 &BOTR, 15–21
 &FIRST, 15–18
 &MIDC, 15–20
 &MIDL, 15–20
 &MIDR, 15–21
 &NONE, 15–18
 &SECOND, 15–18
 &TOPC, 15–20
 &TOPL, 15–20
 &TOPR, 15–20

GPA

Active Part Status, 10–28
 Decimal Places, 10–28
 Symbols, 10–24
 Unit of Measurement, 10–30
 Work Layer, 10–29
 Work View, 10–29

GPA Symbol List, G–3

GPOS, Indicate Generic Point Position, 6–13

GRADE, 1–3

GRADE Program Development, 1–3

Graphic Image of a Matrix, 11–3

GRISEA file, 13–7

GRIP Line Structure, 2–3

GRIP Program Organization, 2–2

GRIPSW, 2–23

GRIPSW/DECLRV, 2–23

I

IF Boolean Operators, 5–21

IF, Logical IF, 5–12

Import an Existing Part, 14–13

Import an Existing Part Grouped, 14–16

INDENT, Select Objects, 7–3

Indicate Generic Point Position, 6–13

Indicate Screen Position Point, 6–11

INEXTE, Initialize Data Model Cycling, 9–11

INEXTN, Initialize Non–geometric Object Cycling,
9–13

Interactive Commands, 6–2, 7–2

Interactive statements, write to dialog box, 6–3

INTERS, 8–15

INTF, 12–5

ISTR, 12–15

J

JUMP, 5–7

Conditional Branching, 5–7

Unconditional Branching, 5–6

L

LABEL, 15–25

LAYER, 10–3

Layer, delete category, 10–8

LAYER, Layer Control, 10–3

LDIM, 15–8

LENF, 12–14

Line, 3–11

Between Two Existing Points, 3–13

Between Two Specified Points (Coordinates),
3–12

EDA, 3–22

Parallel at a Distance, 3–14

Point Tangent to a Curve, 3–18

Through a Point at an Angle, 3–16

Through a Point, Parallel/Perpendicular to a Line,
3–20

Linear Centerline, 15–27

Linking, 1–9

LOGF, 12–5

M

Major Word List, G–1

Major words

ABSF, 12–5



ACOSF, 12–8
ADIM, 15–11
ASATT, 16–9
ASINF, 12–8
ATANF, 12–8
BLEND, 8–16
CAT, 10–5
CATD, 10–8
CATE, 10–7, 10–9
CLINE, 15–27
COSF, 12–7
CREATE, 9–4
DELETE, 10–23
DELNAM, 16–8
DLATT, 16–19
DRAW, 10–20
DRAWE, 14–6
DRAWV, 14–7
ENUM, 16–5
EXPF, 12–5
FDEL, 9–8
FETCH, 9–3
FILE, 9–5
FTERM, 9–6
GRIPSW, 2–23
INEXTN, 9–13
INTERS, 8–15
INTE, 12–5
LABEL, 15–25
LAYER, 10–3
LDIM, 15–8
LOGF, 12–5
MASK, 10–12
MATRIX, 11–6
MAXF, 12–6
MESSG, 6–3
MINF, 12–6
MODE, 12–6
NEXTE, 9–12
NEXTN, 9–15
NOTE, 15–23
OBTAIN, 10–17
RDIM, 15–13
READ, 9–19
RESET, 9–18
RPATT, 14–13
RPATTG, 14–16
SINF, 12–7
SOLBLK, 8–3
SOLCYL, 8–5
SOLEXT, 8–7
SOLREV, 8–9
SQRTF, 12–5
SUBTRA, 8–14
TRANSF, 11–4

TYPEF, 10–15
UNITE, 8–13
WRITE, 9–21
MASK, 10–12
MASK, Class Selection, 10–12
Mathematical Functions, 12–3
MATRIX
 Mirror, 11–8
 Multiple Transformations (Concatenation), 11–10
 Rotate, 11–9
 Scale, 11–7
 Translate, 11–6
MAXF, 12–6
MCHOOS, Choose Multiple Options, 7–15
MESSG, 6–3
MINF, 12–6
Minor Word List, G–2
Mirror, 11–8
MODE, 12–6
Multiple Transformations (Concatenation), 11–10

N

NAME – EDA, object name, 16–3
Name display location (x,y,z), 16–22
Nesting, 4–23
NEXTE, Cycle to Next Object, 9–12
NEXTN, Cycle to Next Non–geometric Object,
 9–15
NOTE, 15–23
Number of Characters in a String, 12–14
Numeric Variables, 2–7

O

Object attributes
 assign, 16–9
 delete name, 16–8
Object Cycling, 9–9
Object name, 16–3
Object Names, 16–2
OBTAIN, Object Information, 10–17
ORIGIN, 15–29

P

PARAM, Enter Parameters, 6–15

Point, 3–4
 Center of Circle, 3–6
 Coordinates, 3–5
 EDA, 3–10
 End Point, 3–7
 Intersection Point, 3–8

POS, Indicate Screen Position Point, 6–11

Position of Characters in a String, 12–23

Positional Modifiers, 3–2

PRINT, Print Data on Listing Device, 9–19, 9–21, 9–22

PROC, 13–4

Procedure Statement, 13–4

Processing Arithmetic Expressions, 12–2

R

RDIM, 15–13

READ, 9–19

Reference String, conversion specifier, 16–10

Replace Characters in a String, 12–20

REPSTR, 12–20

RESET, 9–18

RETURN, 13–6

Return Current Date, 12–10

Return Current Time, 12–12

Return Statement, 13–6

Return String with ASCII Value of N, 12–19

Rotate, 11–9

RPATT, 14–13

RPATT, Import an Existing Part, 14–13

RPATTG, 14–16

RPATTG, Import an Existing Part Grouped, 14–16

S

Scale, 11–7

Select Identities, 7–3

SINE, 12–7

SOLBLK, 8–3

SOLCYL, 8–5

SOLEXT, 8–7

Solid creation
 block, 8–3
 extruded solid, 8–7
 solid cylinder, 8–5
 solid of revolution, 8–9

Solid Feature Creation, 8–2

Solid operations, 8–12
 blend/chamfer, 8–16
 intersect, 8–15
 subtract, 8–14
 unite, 8–13

SOLREV, 8–9

SQRTF, 12–5

String Functions, 12–9

String Variables, 2–13

Subrange Operators, 2–20

Subroutine Programming, 13–2

SUBSTR, 12–22

SUBTRA, 8–14

Suggested Labeling Guidelines, 5–3

T

Text and Arrow Location, 15–19

TEXT, Enter Text, 7–8

TIME, 12–12

TRANSF, Transformation, 11–4

Translate, 11–6

Trigonometric Functions, 12–7

TXARR, 15–19

TYPF, Object Type, 10–15

U

Undeclared Variables, detect, 2–23

UNITE, 8–13

V

VALF, 12–17

Variables, detect undeclared, 2–23

W

WRITE, 9–21



X

XLINE, 15–18

IN

Reference Chart Tear Outs

These tear out reference charts are provided for your convenience.

(This Page Intentionally Left Blank)



STUDENT PROFILE

In order to stay in tune with our customers we are requesting that you take a little time to answer these questions. This information will be kept confidential, and will not be shared with anyone outside of Education Services. PLM Solutions Education Services thanks you for your participation and hopes your training experience will be an outstanding one. Please print the following:

Your Name _____ **Date** _____

Course Number _____ **Course Title** _____

Employer _____

When is your planned departure time? _____ **U.S. citizen?** YES NO

1. Please select the job that best describes your present one.

<input type="checkbox"/> Designer MFG	<input type="checkbox"/> Designer Product	<input type="checkbox"/> Designer Tooling	<input type="checkbox"/> Quality Engineer
<input type="checkbox"/> Engineer MFG	<input type="checkbox"/> Engineer Product	<input type="checkbox"/> Engineer Tooling	<input type="checkbox"/> Process Planner
<input type="checkbox"/> NC Dies	<input type="checkbox"/> NC Other	<input type="checkbox"/> NC Tooling	<input type="checkbox"/> Supplier Quality
<input type="checkbox"/> Drafter MFG	<input type="checkbox"/> Drafter Product	<input type="checkbox"/> Drafter Tooling	<input type="checkbox"/> Workflow Admin.
<input type="checkbox"/> Manager Design	<input type="checkbox"/> Manager Engineering	<input type="checkbox"/> Manager Shop Floor	<input type="checkbox"/> Collaboration Admin.
<input type="checkbox"/> Sys. Admin Local Network	<input type="checkbox"/> Sys. Admin Enterprise	<input type="checkbox"/> Systems Manager	<input type="checkbox"/> Configuration Analyst

2. Your job responsibilities are: _____

3. Reason for training _____

4. What other CAD/CAM/CAE /PDM software have you used? _____

5. What types of parts/data do you model/analyze or store? _____

6. Have you ever had any other instructor–lead/ on–line or self–paced classes for the following:

Subject	From Whom	When	Course Name
<input type="checkbox"/> Unigraphics			
<input type="checkbox"/> I–deas			
<input type="checkbox"/> Imageware			
<input type="checkbox"/> Teamcenter Manufacturing			
<input type="checkbox"/> Teamcenter Engineering (I–Man)			
<input type="checkbox"/> Team Center Enterprise (Metaphase)			
<input type="checkbox"/> Dimensional Management / Visualization			

Unigraphics NX 2 Grip Fundamentals

Course Agenda

Day 1 Monday Morning

- Introduction
- Lesson 1. Grip Development Process
- Lesson 2. Grip Language Components
- Lesson 3 Two Dimensional Commands

Afternoon

- *Assignment 1* *Lines*
- Lesson 4 More Two Dimensional Commands
- *Assignment 2* *Circles*
- Lesson 5 Controlling Program Execution
- *Assignment 3* *Loops*

Day 2 Tuesday Morning

- Lesson 6 Basic Interactive Commands
- *Assignment 4* *Param*
- Lesson 7 Advanced Interactive Commands

Afternoon

- *Assignment 5* *Choose*
- Lesson 8 Solid Object Modeling Commands
- *Assignment 6* *Solids*
- Lesson 9 Database Cycling

Day 3 Wednesday Morning

- *Assignment 7* *Database Cycle*
- Lesson 10 Object Access
- Lesson 11 Transformations

Afternoon

- *Assignment 8* *Transformations*
- Lesson 12 Functions
- *Assignment 9* *Convert Date*

Day 4 _____ Thursday Morning

- *Assignment 9 Convert Date (cont.)*
- Lesson 13 Subroutines
- *Assignment 10 Subroutines*

Afternoon

- *Assignment 10 Subroutines (cont.)*
- Lesson 14 Drafting Functions
- Lesson 15 Dimensions & Drafting Aids

Day 5 _____ Friday Morning

- Lesson 15 Dimensions & Drafting Aids (cont.)
- *Assignment 11 Dimensions*
- Lesson 16 Attributes (Optional)
- Finish Final Project



Level 1 Evaluation

Course Code _____ Course _____ Date _____ Instructor _____

1. STRONGLY DISAGREE	2. DISAGREE	3. SOMEWHAT AGREE	4. AGREE	5. STRONGLY AGREE
----------------------	-------------	-------------------	----------	-------------------

Content:

1. The course objectives were clear _____
3. The goals and objectives for this course were met _____
4. The course flowed in a logical and meaningful manner _____
5. The course was the appropriate length for the info taught _____

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Materials:

1. The training materials supported the course objectives _____
2. The training materials were logically sequenced _____
3. Enough information was provided to complete workshops _____
4. The course materials provided clear and descriptive directions _____
5. The exercises and workshops supported the learning experience _____
6. The materials were easy to read and understand _____

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Instructor:

1. The instructor was knowledgeable about the subject _____
2. He/She answered my questions appropriately _____
3. He/She was well prepared to deliver the course _____
4. He/She was organized and well spoken _____
5. The instructor encouraged questions in class _____
6. The instructor made good use of the training time _____
7. The instructor completed the course _____
8. The instructor used examples relevant to the course _____
9. The instructor provided enough time to complete the exercises _____
10. The instructor used review tests and gave feedback _____

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Student Self-Evaluation:

1. I met the prerequisites for the class (I had the skills I needed) _____
2. I will be able to use the skills I have learned on my job _____
3. My expectations for this course were met _____
4. I am confident that with practice I will become proficient _____

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

Facilities:

1. The computer equipment was reliable _____
2. The software performed properly _____
3. The overhead projection unit was clear and working properly _____
4. The registration and confirmation process was efficient _____
5. The training facilities were comfortable and clean and provided a good learning environment _____

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

(Over)

Hotels: (Optional)

1. Was this hotel recommended during your registration process? _____ YES NO

If not how was it chosen _____

2. Was the hotel clean? _____ YES NO

3. Did it have a full-serve restaurant? _____ YES NO

4. Overall impression of the Hotel _____

1	2	3	4	5
---	---	---	---	---

5. What was the name of the hotel? _____

Please check this box if you would like your comments below featured in our training publications.
(Your name is required at the bottom of this form)

Please check this box if you would like to receive more information on our other courses and services.
(Your name is required at the bottom of this form)

We would appreciate it if you would take a minute to add some personal comments on this form. These comments help us to better understand the ever-changing student needs.

Thank you for choosing EDS PLM Education Services, and we hope we can be of help to you in the future.

Comments: _____

Name Optional: _____ Location _____

Software Version _____

