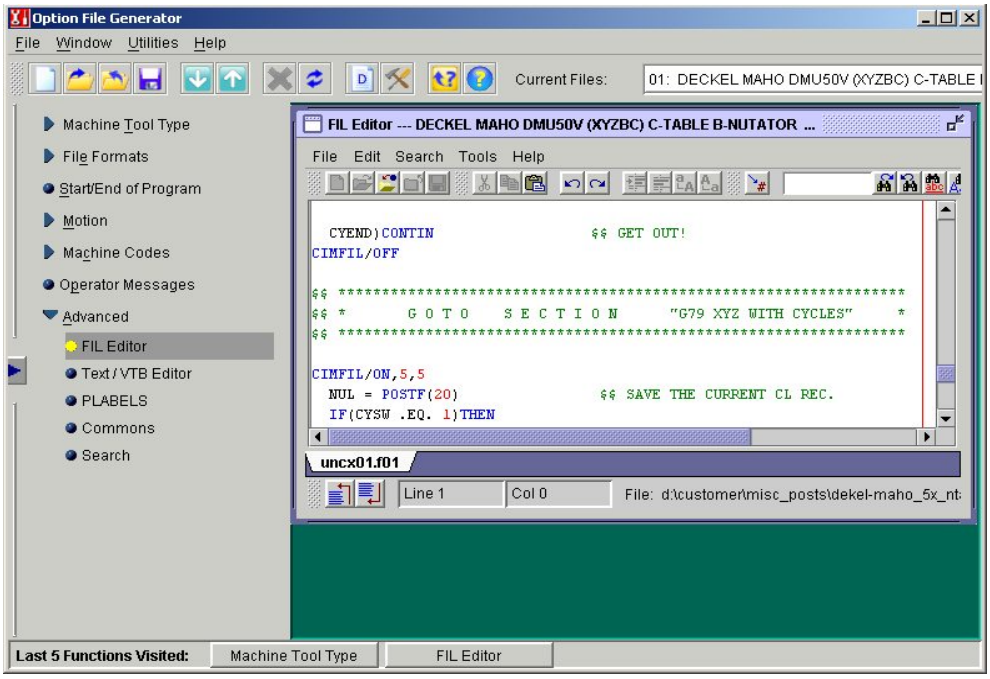
FIL



Factory Interface Language

FIL *Version 6.1*

Austin N.C., Inc. Technical Assistance

Austin N.C., Inc.'s products are completely supported via our **Continuous Support Program** (**CSP**). Your **CSP** contract includes technical assistance and software updates. For assistance with this software product, or for information about your CSP maintenance contract, contact Austin N.C., Inc. by one of the following methods:

Telephone:.....	(512) 458-1112
FAX:.....	(512) 458-5474
Email:.....	support@austinnc.com
Web Site:.....	http://www.austinnc.com

Austin N.C., Inc.
Dba Intercim-Austin
505 East Huntland Drive, Suite 480
Austin, Texas 78752
http://www.austinnc.com

LIMITATIONS OF WARRANTY AND LIABILITY

Austin N.C., Inc. makes no warranty, expressed or implied, with respect to this manual, the accompanying software, and any other related items, their quality, performance merchantability, or fitness for any particular use. It is solely the purchaser's responsibility to determine their suitability for any particular purpose. Austin N.C., Inc. will in no event be held liable for direct, indirect, or incidental damages resulting from any defect or omission in the software or other related items and processes, including but not limited to any interruption of services, loss of business or anticipatory profit, or other consequential damages. This statement of limited liability is in lieu of all other warranties or guarantees, expressed or implied, including warranties of merchantability and fitness for a particular purpose. Austin N.C., Inc. neither assumes nor authorizes any person to assume for it any other warranty or liability in connection with the sale of its products.

Product Improvements

The information in this document is subject to change without notice and should not be construed as a commitment by Austin N.C., Inc. Austin N.C., Inc. assumes no responsibility for any errors that may appear in this document.

Factory Interface Language (FIL) ™ Manual
Version 6.1

November 30, 2006

Notice of Trademarks

MAN-FIL

Read This First!

If you are one of those people who likes to read software manuals from cover to cover in one sitting, you can skip this section. However, if you prefer to read only the portions of the manual that explain the tasks you want to do, you can save time by reading these pages before you go any further.

Chapter 1 begins with an explanation of how this manual is set up. You can skip that if you want; you probably are familiar with our books anyway. However, be sure to read section 1.2, **First, a Few Words...** That section tells you what you need to know and what you need to have before you begin to use FIL.

Chapter 2 is an overview of FIL. You need to read that chapter, too. (It's short, and won't take you long.)

Chapter 3 explains the CL record structure. If you are not familiar with that structure, you need to read this chapter pretty carefully, If you are familiar with CL records, you might want to glance at the pictures. (Of course, we would not be offended if you read the words, too.)

Chapter 4 explains the FIL command language, the file syntax, the commands and functions that are available to you. You need to read this chapter.

Chapter 5 explains the FIL **POSTF** (post function) commands. These commands are the heart and soul of FIL. You need to read this chapter, also.

That is really all the required reading. Chapter 6 is filled with some practical examples of FIL coding for your enlightenment and enjoyment.

Chapter 7 is the G-Post integer code reference section. It's there for your convenience.

Chapter 8 **REPLAC** MCD file output text replacement

.

Chapter 9 _**MCDWT** Macro used for editing the **MCD block**.

Chapter 10 _**OUTPT** Macro used for editing the **OUTPUT buffer**.

Chapter 11 _**REPOS** Macro used for automatic rotary axis reposition after an axis limit has been violated.

Chapter 12 Interactive Debugger used for assisting the users in developing post processors.

We have put a detailed index at the end of the manual so you can find what you need, quickly.

We hope that after you read the manual, you will let us know what you liked or disliked about it. We encourage your comments as they help us to improve our products. (Besides, how often do you get a chance to tell *us* what to do?)

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Have you looked over the section entitled **Read This First!** at the front of the manual? Now would be a good time to do so, especially if you do not intend to read the entire manual in one sitting.

This chapter provides an overview of the Austin N.C., Inc. Factory Interface Language (FIL). It begins with a discussion of the conventions used throughout the manual, then goes on to tell you what you need to know to use FIL.

This chapter is required reading. Section 1.2 tells you some things you must know before you go any further.

1.1 Manual Conventions

We use certain conventions to present information in this manual. This section tells you what they are.

1.1.1 Notes, Cautions, and Warnings

A **Note** is information that is of interest or importance, such as the following:

Note:	**Some machines seem to know when the schedule is tight and so choose only those times to break down. During one of those down-times, we suggest that you relax, enjoy this manual, and send us your suggestions for improving it.**

A **Caution** contains very important information that you need to know so you will not lose or corrupt data.

Caution:	**If you turn off the computer without saving your file, you will lose the data that you entered.**

A **Warning** is information you need to know to avoid injury to a person or damage to equipment.

Warning:	**If you bypass the safety device, the machine might overheat.**

1.1.2 Examples

The manual frequently contains examples to aid in the explanation of various commands. Those examples are shown in the following typeface:

GOTO/0,0

Programming examples are shown in the following typeface:

SET/START,XAXIS,0,YAXIS,0

Command or option names are shown in all capital letters, boldface type:

MACHIN/*statement*.

Examples of text or data that you enter are shown in a different boldface type:

PARTNO THIS IS A TEST DATA STATEMENT

File names are shown in the following typeface:

FILENAME.EXT.

1.2 First, a Few Words...

Before we explain FIL, you need to know what assumptions we are making about you and the experience you have. ***It is vital that you read and understand this section***.

1.2.1 What You Need to Know

FIL is a powerful product that will do almost anything you tell it to do during the post processing phase. It is important that you understand the implications of the commands you give FIL. We strongly suggest you have a better-than-average working knowledge of the following:

- Logic capabilities of FIL
- FIL syntax
- CL file structure
- Post Processors

You must also attend the FIL training classes given by Austin N.C., Inc..

1.2.2 What You Need to Have

You must have the following to use FIL:

- Austin N.C., Inc. Lathe G-Post *or* Mill G-Post software.
- Austin N.C., Inc. CAD/CAM interface (XPost)

This manual refers to other Austin N.C., Inc. manuals in several places. You will need the following manuals:

- G-Post Generalized Post Processor Reference Manual
- CIMpro NC System Guide

If you do not already have these manuals, you can order them from Austin N.C., Inc. at the following address:

Austin N.C., Inc.
Dba: Intercim-Austin
505 East Huntland Drive, Suite 480
Austin, Texas 78752

2 Overview

2.1 Introduction

This chapter provides a broad overview of the Factory Interface Language. We explain the basics of how to use FIL and tell you where to look for specific reference information.

Please read this chapter carefully.

2.2 Overview of the Factory Interface Language

FIL enables you to do the following tasks:

- Add, delete, or modify CL file data
- Alter post processor output
- Add or modify post processor vocabulary words
- Read or write data in up to two files
- Call other applications

You do this by writing a filter file called ***UNCL01.Fnn*** (if you are using the Lathe G-Post) or ***UNCX01.Fnn*** (if you are using the Mill G-Post). The ***nn*** in the file name represents the machine number you used with the option file you already have. For example, to use FIL with the Lathe G-Post, you would have an option file ***UNCL01.Pnn*** and a filter file ***UNCL01.Fnn***. If you need help creating an option file, see the G-Post Generalized Post Processor Reference Manual.

FIL inserts data into the post through the filter file. You can program any of the FIL syntax commands in the filter file (see Chapter 4).

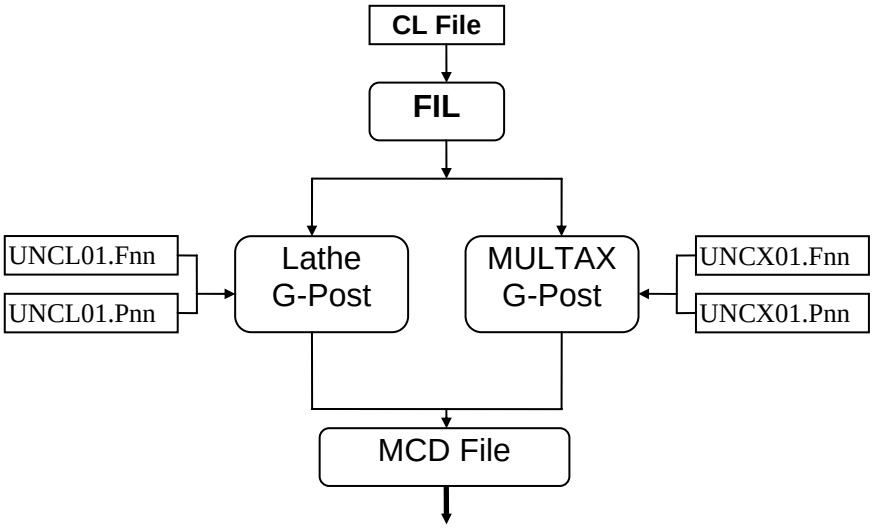


Figure 2-1. Factory Interface Language

FIL is a powerful addition to the Austin N.C., Inc. G-Post. As with any high-powered tool, you must be careful when you use FIL. We suggest that you have experience with post processing and that you understand the implications of modifying a CL file before you use FIL.

2.3 First Things First: Plan

Before you write the FIL file, we suggest that you plan your approach carefully. This consists of three steps:

1. Define the problem.
2. Approach the solution logically.
3. Examine the implications carefully.

If you bypass these steps, you run the risk of creating a real problem.

Many times you'll find that the process of defining the problem leads you to the solution. Because FIL lets you do just about anything you want to, it's vital that you think through the implications of what you plan to do.

Warning:	**It is possible to modify your post processor output in ways that might result in damage to your machines. Verify your FIL code carefully before you implement it.**

2.4 The FIL File

Use the Option File Generator FIL Editor or any ASCII editor to write the filter file. There are two rules regarding the filter file name:

- The file name must correspond to the G-Post you are using. If you are using the Lathe G-Post, you must call the file ***UNCL01.Fnn*****.** If you are using the MULTAX (Mill) G-Post, you must call the file ***UNCX01.Fnn***.

- The file extension must correspond with the machine number you use with the option file. For example, if your option file is called ***UNCX01.P01***, you must call the FIL file ***UNCX01.F01***.

The standard search rules are in effect if the filter file is not in the current directory, FIL looks in the **UNC$LIBRARY** directory.

2.5 Syntax

Before reading the first CL record, FIL performs a syntax check on the filter (FIL) file. If there are no errors, the file processes normally. If there are errors, the file processes, but you might get unexpected results in the post output. Be sure to review the Listing File (.LST) for possible error messages. Because this is important, we're going to repeat it:

Warning:	**If the syntax check finds errors in the filter file, the file processes. However, you might get unexpected results in the post output. Be sure to review the Listing File (.LST) for possible error messages.**

After the syntax check, FIL performs a logical execution of everything outside the FIL routines. (A FIL routine is the information between the **CIMFIL/ON-AT-ALL-AUTO** and **CIMFIL/OFF** statements.) This enables you to set global variables up front, outside the FIL routines. If you use macros, we suggest that you define them first. A macro cannot be defined within a FIL routine.

The following example illustrates this:

```
PRINT/ON        )     Statements outside the FIL routines
FIRST=0         |     are executed once.
RESERV/A,10     |
M1=MACRO        |
—               |
TERMAC          )

CIMFIL/ON,SPINDL    )     $$ Begin FIL routine
  RSLT=POSTF(13)    |     Statements checked for syntax at
  .                 |     first. Only executed when a
  .                 |     SPINDL record is encountered
  .                 |     in the CL file.
CIMFIL/OFF          )     $$ End FIL routine
```

There are two basic types of FIL routines. Those which process all CL records of a particular ***Type/Subtype*** (i.e. **CIMFIL/ON**) and those which process only select CL record of a particular ***Type/Subtype*** (i.e. **CIMFIL/AT-AUTO-ALL**).

2.5.1 Defining FIL routines to capture all CL records of a particular Type/Subtype

Using **CIMFIL/ON** to select a CL record only checks the ***Type*** and ***Subtype*** of the CL record. This means that you will get every CL record of the desired ***Type*** and ***Subtype***. **CIMFIL/ON** is the preferred method of catching CL records as it allows you to analyze every CL record of the desired ***Type*** and ***Subtype***.

2.5.1.1 CIMFIL/ON,*major_word*

This specifies the beginning of a FIL routine for the defined ***major word.*** All CL records of this type will be processed by this FIL routine as they are encountered unless a **CIMFIL/AT** or **CIMFIL/AUTO** or **CIMFIL/ALL** is defined for the same ***major word.***

major_word is any Post Processor Type-2 major word in the CL file. In addition to the Type 2 major words, the following **CIMFIL/ON** syntax applies:

CIMFIL/ON,CUTTER same as CIMFIL/ON,6,6
CIMFIL/ON,FROM same as CIMFIL/ON,5,3
CIMFIL/ON,GOTO same as CIMFIL/ON,5,5
CIMFIL/ON,CIRCLE same as CIMFIL/ON,3
CIMFIL/ON,MULTAX same as CIMFIL/ON,9
CIMFIL/ON,FINI same as CIMFIL/ON,14
CIMFIL/ON,REMARK same as CIMFIL/ON,2,1042

2.5.1.2 CIMFIL/ON,Type[,Subtype]

This is an alternate format of the above **CIMFIL/ON,*major_word.*** Instead of the ***major_word,*** you can specify the CL record ***Type[,Subtype].*** For example, **CIMFIL/ON,SPINDL** is the exact same command as **CIMFIL/ON,2,1031**. **CIMFIL/ON,*Type[,Subtype]*** functions exactly the same as the above **CIMFIL/ON,*major_word.***

Type is any CL record type, such as 2 for a post record, 3 for a circular move record, or 5 for a motion record. ***Subtype*** is an optional entry. If you do not use S***ubtype***, the program matches only the record type.

2.5.1.3 CIMFIL/OFF

CIMFIL/OFF terminates the current FIL routine and returns control to the G-Post to process the CL file. Using **CIMFIL/OFF** is optional, since the next **CIMFIL/ON-AT-AUTO-ALL** terminates the current FIL routine.

Note:	**CIMFIL/ON-OFF** commands used in conjunction with other FIL functions (IF, DO, POSTF, etc.) will enable you to capture any post command in the CL file and alter the tape output. **CIMFIL/AT,ALL,AUTO** can be used in place of the above method. You do not need be familiar with this form of **CIMFIL** syntax. Unless you wish to program your FIL code using this form, feel free to skip the next section (2.6).

2.5.2 Defining FIL routines to capture only select CL records of a particular Type/Subtype

Using **CIMFIL/AT-AUTO-ALL** allows you to further specify search criteria for a desired CL record and retrieve only those CL records that match the criteria. For example, instead of processing all SPINDL CL records you can process just the SPINDL/OFF CL records. Using the **CIMFIL/AT-AUTO-ALL** commands you can reduce the size and complexity of each FIL routine. By letting FIL select the desired CL record you will not be required to perform this task yourself, within a **CIMFIL/ON** routine, saving you many lines of FIL code.

A simple rule to follow for deciding which **CIMFIL/command** best fits your needs (**SPINDL** CL records are used as an example):

1. To catch all **SPINDL** CL records.
   Use **CIMFIL/ON,SPINDL**

2. To catch all **SPINDL** CL records with **CLW** at the 4th CL word location..
   Use **CIMFIL/AT,SPINDL,CLW**

3. To catch all **SPINDL** CL records with **CLW** at the 4th CL word location and match the CL record exactly in number of words (length).
   Use **CIMFIL/AUTO,SPINDL,CLW**

4. To catch all **SPINDL** CL records with **CLW** anywhere in the CL record.
   Use **CIMFIL/ALL,SPINDL,CLW**

2.5.2.1 CIMFIL/AT[,SAME-SMALL-LARGE],*major_word,DATA-scalar-word,…*

CIMFIL/AT is an "Exact or Up To" match of the CL record. The CL record will be searched for an exact or up to match of the specified ***DATA-scalar-word*** combinations. Additional CL words in the CL record, following the ***DATA-scalar-word*** criteria, are acceptable.

The ***DATA-scalar-word*** comparison will start at the 4th CL word of the CL record, this is the 1st word to the right of the slash (/) in a command, except for Type 5 (Motion) and Type 3 (Circle) CL records.

Type 5 (Motion CL records); the ***DATA-scalar-word*** comparison will start at the 6th CL word.

Type 3 (Circle CL records); the ***DATA-scalar-word*** comparison will start at the 9th CL word.

When searching for ***scalar*** values in the CL Record there are three types of comparisons.

SAME specifies "Equal To" (±0.000001 is considered equal).
SMALL specifies "Less Than"
LARGE specifies "Greater Than"

SAME is the default comparison and may be specified, but is not required in the syntax.

major_word is any Post Processor Type-2 major word in the CL file. In addition to the Type 2 major words, the following **CIMFIL/AT** syntax applies:

CIMFIL/AT,CUTTER
CIMFIL/AT,FROM
CIMFIL/AT,GOTO
CIMFIL/AT,CIRCLE
CIMFIL/AT,MULTAX
CIMFIL/AT,FINI
CIMFIL/AT,REMARK

DATA is a place holder or wild-card. Use **DATA** when you need to match a CL word that follows a CL word that can vary.

Example:

Let us say you want to process all **SPINDL/*ssss*,RPM** (***ssss*** is the speed desired) CL records. You would use **DATA,** as a wild-card for the 4th CL word, in the **CIMFIL/AT** command.

```
CIMFIL/AT,SPINDL,DATA,RPM
    --
    --
CIMFIL/OFF
```

The following CL record will be processed as they match the search criteria.

```
SPINDL/200,RPM,CLW,RANGE,1
SPINDL/88,RPM,RANGE,1
SPINDL/190,RPM,CLW
```

The following CL record would not be processed by this FIL routine because the RPM is not the 5th CL Word

SPINDL/200,SFM,CLW,RANGE,1

Example:

Let us say you want to process all **LOADTL/*tttt*,ADJUST,*aaaa*,LENGTH,*llll*** (***tttt*** is the tool number, ***aaaa*** is the offset number and ***llll*** is the length value) CL records. You would use **DATA,** as a wild-card for the 4th CL word and 6th CL words, in the **CIMFIL/AT** command.

CIMFIL/AT,LOADTL,DATA,ADJUST,DATA,LENGTH
--
--
CIMFIL/OFF

The following CL records will be processed as they match the search criteria.

LOADTL/1,ADJUST,11,LENGTH,0
LOADTL/2,ADJUST,22,LENGTH,1.5
LOADTL/3,ADJUST,44,LENGTH,4.5
LOADTL/4,ADJUST,44,LENGTH,2.4

The following CL record would not be processed by this FIL routine because the ADJUST is not the 5th CL Word

LOADTL/4,LENGTH,2.4

The following CL record would not be processed by this FIL routine because the LENGTH is not the 7th CL Word

LOADTL/4,ADJUST,5.5

scalar is a numeric value . When you specify a ***scalar***, the specified CL word location must contain the desired value based on the **SAME-SMALL-LARGE** settings for this command.

Example:

Let us say you want to process all **SPINDL/500** CL records. You would use **500,** as a 1st ***scalar*** in the **CIMFIL/AT** command.

CIMFIL/AT,SPINDL,500
--
--
CIMFIL/OFF

The following CL records will be processed as they match the search criteria.

SPINDL/500,RPM,CLW,RANGE,1
SPINDL/500,RPM,RANGE,1
SPINDL/500
SPINDL/500,CLW

The following CL records would not be processed by this FIL routine because the 4TH CL word does not contain **500.**

SPINDL/RPM,500
SPINDL/1500,RPM,CLW,RANGE,1
SPINDL/RPM,1500,RANGE,1

word is a valid minor word (i.e. **RPM, SFM, ADJUST**, etc.). When you specify a ***word***, the specified CL word location must contain the desired ***word***.

Example:

Let us say you want to process all **CUTCOM/LEFT** CL records. You would use **LEFT,** as a 1st ***word*** in the **CIMFIL/AT** command.

CIMFIL/AT,CUTCOM,LEFT
--
--
CIMFIL/OFF

The following CL records will be processed as they match the search criteria.

CUTCOM/LEFT,1
CUTCOM /LEFT,2,XYPLAN
CUTCOM /LEFT

The following CL records would not be processed by this FIL routine because the 4TH CL word does not contain **LEFT.**

CUTCOM /RIGHT,1
CUTCOM /OFF

2.5.2.2 CIMFIL/AUTO[,SAME-SMALL-LARGE],*major_word,DATA-scalar-word,…*

CIMFIL/AUTO is an "Exact" match of the CL record. The CL record will be searched for an exact match of the specified ***DATA-scalar-word*** combinations. Additional CL words in the CL record, following the ***DATA-scalar-word*** criteria, will cause the match to fail.

The ***DATA-scalar-word*** comparison will start at the 4th CL word of the CL record, this is the 1st word to the right of the slash (/) in a command, except for Type 5 (Motion) and Type 3 (Circle) CL records.

Type 5 (Motion CL records); the ***DATA-scalar-word*** comparison will start at the 6th CL word.

Type 3 (Circle CL records); the ***DATA-scalar-word*** comparison will start at the 9th CL word.

When searching for ***scalar*** values in the CL Record there are three types of comparisons.

SAME specifies "Equal To" (±0.000001 is considered equal).
SMALL specifies "Less Than"
LARGE specifies "Greater Than"

SAME is the default comparison and may be specified, but is not required in the syntax.

major_word is any Post Processor Type-2 major word in the CL file. In addition to the Type 2 major words, the following **CIMFIL/AUTO** syntax applies:

CIMFIL/AUTO,CUTTER
CIMFIL/ AUTO,FROM
CIMFIL/ AUTO,GOTO
CIMFIL/ AUTO,CIRCLE
CIMFIL/ AUTO,MULTAX
CIMFIL/ AUTO,FINI
CIMFIL/ AUTO,REMARK

DATA is a place holder or wild-card. Use **DATA** when you need an exact match of a CL word that follows a CL word that can vary.

Example:

Let us say you want to process all **SPINDL/*ssss*,RPM** (***ssss*** is the speed desired) CL records. You would use **DATA,** as a wild-card for the 4th CL word, in the **CIMFIL/AUTO** command.

CIMFIL/AUTO,SPINDL,DATA,RPM
--
--
CIMFIL/OFF

The following CL records will be processed as they match exactly.

SPINDL/88,RPM
SPINDL/190,RPM

The following CL records would not be processed by this FIL routine because they do not match the search criteria exactly

SPINDL/200,RPM,CLW,RANGE,1
SPINDL/200,SFM,CLW,RANGE,1
SPINDL/200,SFM
SPINDL/200,RPM,CLW
SPINDL/200,RPM,RANGE,1

Example:

Let us say you want to catch all **LOADTL/*tttt*,ADJUST,*aaaa*,LENGTH,*llll*** (***tttt*** is the tool number, ***aaaa*** is the offset number and ***llll*** is the length value) CL records. You would use **DATA,** as a wild-card for the 4th CL word and 6th CL words, in the **CIMFIL/AUTO** command.

CIMFIL/AUTO,LOADTL,DATA,ADJUST,DATA,LENGTH,DATA
--
--
CIMFIL/OFF

The following CL records will be processed as they match exactly.

LOADTL/1,ADJUST,11,LENGTH,0
LOADTL/2,ADJUST,22,LENGTH,1.5
LOADTL/3,ADJUST,44,LENGTH,4.5
LOADTL/4,ADJUST,44,LENGTH,2.4

The following CL records would not be processed by this FIL routine because they do not match the search criteria exactly

LOADTL/4,LENGTH,2.4
LOADTL/4,ADJUST,5.5

scalar is a numeric value . When you specify a ***scalar***, the specified CL word location must contain the desired value based on the **SAME-SMALL-LARGE** settings for this command.

Example:

Let us say you want to process all **SPINDL/500** CL records. You would use **500,** as a 1st ***scalar*** in the **CIMFIL/AUTO** command.

CIMFIL/AUTO,SPINDL,500
--
--
CIMFIL/OFF

The following CL record will be processed as it matched exactly.

SPINDL/500

The following CL records would not be processed by this FIL routine because they do not match the search criteria exactly

SPINDL/500,CLW
SPINDL/500,RPM,CLW,RANGE,1
SPINDL/500,RPM,RANGE,1
SPINDL/RPM,500
SPINDL/1500,RPM,CLW,RANGE,1
SPINDL/RPM,1500,RANGE,1

word is a valid minor word (i.e. **RPM, SFM, ADJUST**, etc.). When you specify a ***word***, the specified CL word location must contain the desired ***word***.

Example:

Let us say you want to catch all **CUTCOM/LEFT** CL records. You would use **LEFT,** as a 1st ***word*** in the **CIMFIL/AUTO** command.

CIMFIL/AUTO,CUTCOM,LEFT
--
--
CIMFIL/OFF

The following CL record will be processed as it matched exactly.

CUTCOM /LEFT

The following CL records would not be processed by this FIL routine because they do not match the search criteria exactly

CUTCOM/LEFT,1
CUTCOM /LEFT,2,XYPLAN
CUTCOM /RIGHT,1
CUTCOM /OFF

2.5.2.3 CIMFIL/ALL,*major_word,scalar-word,…*

CIMFIL/ALL is used to match any ***scalar*** or ***word*** in the CL record. Unlike **CIMFIL/AT** or **CIMFIL/AUTO** the position is not matched. If there is a match then the CL record will be processed using this FIL routine.

major_word is any Post Processor Type-2 major word in the CL file. In addition to the Type 2 major words, the following **CIMFIL/ALL** syntax applies:

CIMFIL/ALL,CUTTER
CIMFIL/ ALL,FROM
CIMFIL/ ALL,GOTO
CIMFIL/ ALL,CIRCLE
CIMFIL/ ALL,MULTAX
CIMFIL/ ALL,FINI
CIMFIL/ ALL,REMARK

scalar is a numeric value . When you specify a ***scalar***, if that ***scalar*** is found anywhere in the CL record the CL record will be processed though this FIL routine.

Example:

Let us say you want to catch all **SPINDL/** CL records that contain **500**. You would use **500,** as the ***scalar*** in the **CIMFIL/ALL** command.

CIMFIL/ALL,SPINDL,500
--
--
CIMFIL/OFF

The following CL records will be processed as they contain **500**.

SPINDL/500
SPINDL/500,CLW
SPINDL/500,RPM,CLW,RANGE,1
SPINDL/500,RPM,RANGE,1
SPINDL/RPM,500
SPINDL/500,RPM,CLW,RANGE,1
SPINDL/RPM,500,RANGE,1
SPINDL/SFM,100,RANGE,1,MAXRPM,500

The following CL records would not be processed by this FIL routine because they do not have a scalar 500 anywhere.

SPINDL/RPM,1500
SPINDL/OFF
SPINDL/ON
SPINDL/RPM,1500,RANGE,1,MAXRPM,5500

word is a valid minor word (i.e. **RPM, SFM, ADJUST**, etc.).When you specify a ***word***, if that ***word*** is found anywhere in the CL record, the CL record will be processed though this FIL routine..

Example:

Let us say you want to process all **CUTCOM/** CL records that contain **OFF**. You would use **OFF** as the ***word*** in the **CIMFIL/ALL** command.

CIMFIL/ALL,CUTCOM,OFF
--
--
CIMFIL/OFF

The following CL records will be processed as they contain **OFF**.

CUTCOM /OFF
CUTCOM /LEFT,OFF
CUTCOM /RIGHT,OFF

The following CL records would not be processed by this FIL routine because they do not contain **OFF**.

CUTCOM/LEFT,1
CUTCOM /LEFT,2,XYPLAN
CUTCOM /RIGHT,1

Notes:

1. You can have multiple **CIMFIL/AT-AUTO-ALL** commands to catch and apply desired FIL exceptions.

2. If more than one **CIMFIL/ON-AT-AUTO-ALL** are found in the FIL file for the same major word (say **SPINDL**), then the G-Post will use the following search rule:

First, search the **CIMFIL/AUTO** table.
If no match, search the **CIMFIL/AT** table.
If no match, search the **CIMFIL/ALL** table.
If no match, search the **CIMFIL/ON** table.

3. If you use both **CIMFIL/AT-AUTO,SMALL-LARGE** and **CIMFIL/AT-AUTO,SAME** format then **CIMFIL/AT-AUTO,SAME** will be matched first, whether the word **SAME** is given or implied.

Example:

Current CL record is **GOTO/2,1,3**

Your FIL is:

```
CIMFIL/AT,SMALL,GOTO,3          $$ CATCH X<3
      --
CIMFIL/OFF

CIMFIL/AT,GOTO,2                $$ CATCH X=2
      --
CIMIFL/OFF
```

Since the CL record matches both test, the equal condition is matched even though it is defined after the **CIMFIL/AT,SMALL,GOTO,3** in the FIL file.

4. If you use multiple **CIMFIL/AT-AUTO,SMALL-LARGE** sections, the best matching pattern for the CL record will be chosen.

Example:

Current CL record is **GOTO/2,1,3**

Your FIL is:

```
CIMFIL/AT,SMALL,GOTO,3          $$ CATCH X<3
      --
CIMFIL/OFF

CIMFIL/AT,SMALL,GOTO,2.5        $$ CATCH X<2.5
      --
CIMIFL/OFF
```

Since the CL record matches both test, the smallest condition is matched even though it is defined after the **CIMFIL/AT,SMALL,GOTO,3** in the FIL file.

5. You can combine these commands and use the new **POSTF(28-29)** functions to quickly parse the CL records.

6. The ***DATA-scalar-word*** comparison will start at the 4th CL word of the CL record, this is the 1st word to the right of the slash (/) in a command, except for Type 5 (Motion) and Type 3 (Circle) CL records.

Type 5 (Motion CL records); the ***DATA-scalar-word*** comparison will start at the 6th CL word.

Type 3 (Circle CL records); the ***DATA-scalar-word*** comparison will start at the 9th CL word.

7. Do not use **DATA** for **CIMFIL/ALL** formats as it is not matched by position.

8. You must supply at least one ***scalar-word*** after the ***major word*** with **CIMFIL/AT-AUTO** or **CIMFIL/ALL**. If not an error #108 will be generated.

9. If you use multiple **CIMFIL/ALL** commands and more than one causes a CL record match, the first one defined in the FIL file will be selected. This is one of the few cases that order of definition in the FIL file makes a difference.

Example:

Let us say we have two **CIMFIL/ALL** commands in our FIL file;

```
CIMFIL/ALL,SPINDL,CLW          $$ check SPINDL for CLW
        --
        --
CIMFIL/OFF

CIMFIL/ALL,SPINDL,RANGE        $$ check SPINDL for RANGE
        --
        --
CIMFIL/OFF
```

And the current CL record is;

```
SPINDL/500,RPM,CLW,RANGE,2
```

In this case, both of the **CIMFIL/ALL** FIL routines are valid for this CL record. Since both are valid we will always choose the 1st one defined, in this case **CIMFIL/ALL,SPINDL,CLW** will be selected. and **CIMFIL/ALL,SPINDL,RANGE** will be ignored.

10. If you define two or more identical **CIMFIL/ON** commands. for a CL record match, the first one defined in the FIL file will be selected. This is one of the few cases that order of definition in the FIL file makes a difference.

Example:

Let us say we have two **CIMFIL/ON,LOADTL** commands in our FIL file;

```
CIMFIL/ON,LOADTL               $$ Catch all LOADTL CL records
        --
        --
CIMFIL/OFF

CIMFIL/ON,LOADTL               $$ Catch all LOADTL CL records
        --
        --
CIMFIL/OFF
```

And the current CL record is;

```
LOADTL/1,ADJUST,1
```

In this case, you defined two identical **CIMFIL/ON** FIL routines. Since both are valid we will always choose the 1st one defined, this is normally done by mistake and should be avoided as it could lead to difficulty in debugging.

2.6 FIL Command and Syntax

Chapter 4 explains the commands and syntax developed specifically for use with FIL. Please read that chapter carefully.

2.7 The POSTF Function

The **POSTF** function is the portion of FIL that enables you to access CL file data, including post processor COMMON variables. Chapter 5 explains the options available to you with the **POSTF** function.

Refer to the *G-Post Generalized Post Processor Reference Manual* for COMMON variable locations.

2.8 Some Final Thoughts

FIL is an interpretive interface language. Use small sections, and resolve one section at a time. Although you can have as many FIL blocks as you want, you'll have best results when you keep the FIL code small and simple.

Be sure to document the code internally so that anyone can interpret the intent. Of course, *we* never forget why we coded something a certain way. The reason we document our code is so that anyone who might come along later can marvel at our genius. But we certainly appreciate it when we work with code that the guy down the hall was thoughtful enough to document.

3 CL Record Format

Introduction

This chapter shows the logical CL record format for each CL record type. You'll need this information to understand the **POSTF** function described in Chapter 5.

3.1 CL Record Structure

The records of the CL file contain both integer and floating-point values (scalars). The ICLWRD array contains the integer code for vocabulary words. The CLWRD array contains scalar values that are passed to the post processor.

When an ICLWRD array location is loaded, a zero is placed in the equivalent CLWRD location. When the CLWRD location is loaded, a zero is placed in the equivalent ICLWRD location. In other words, one location contains data and the other doesn't. (Of course, if both locations contain a zero, it means that a zero was programmed in CLWRD.)

The following example illustrates the loading of a **SPINDL/300,CLW** record:

```
SPINDL:     ICLWRD(3) =1031     CLWRD(3) =  0.0
300.0:      ICLWRD(4) =   0     CLWRD(4) =  300.0
CLW:        ICLWRD(5) =  60     CLWRD(5) =  0.0
```

Figure 3-1 illustrates the CL record structure.

1	2	3	4	5
CL Record Number	**Record Type**	**Major Word**	**Type and Value**	**Type and Value**
Integer	**Integer**	**Integer**	**CLWRD *or* ICLWRD**	**CLWRD *or* ICLWRD**

Figure 3-1. CL Record Structure

The number at the top of each column indicates the location number. The first three locations are always integers; these integers represent the information to the left of the slash in the post processor statement. Locations 4 through *n* represent the information to the right of the slash in the post processor statement; these values are treated like real numbers. Type 1000 records consist of only the first three locations.

Most of the work you do with FIL involves Type 2000 records. To continue the previous example, Figure 3-2 illustrates a **SPINDL/300,CLW** record. Location 4 is the scalar value of the CLWRD array, and location 5 is the vocabulary word integer code of the ICLWRD array.

1	2	3	4	5
CL Record Number	**Record Type**	**SPINDL**	**300**	**CLW**
N	**2000**	**1031**	**300.0**	**60**

Figure 3-2. CL Record Structure Example

3.2 Type 1000 Source Statement Record

A Type 1000 record precedes every CL record except a Type 5000 Subtype 6, continuation record. This provides the number of the CL source statement that was responsible for generating the record.

ICLWRD (1) = CL record number
ICLWRD (2) = 1000
ICLWRD (3) = CL source statement number

Figure 3-3 shows the structure for a Type 1000 CL record.

1	2	3
CL Record Number	**Record Type**	**Major Word**
n	**1000**	***n***

Figure 3-3. CL Record Structure

3.3 Type 2000 Post Processor Command Record

These are the variable assignments for Type 2000 records:

ICLWRD (1) = CL record number
ICLWRD (2) = 2000
ICLWRD (3) = Integer code for command (major word)
ICLWRD (4)
or
CLWRD (4) = First parameter to the right of the slash
ICLWRD (5)
or
CLWRD (5) = Second parameter
.
.
.
ICLWRD (n)
or
CLWRD (n) = nth parameter

1	2	3	4	5
CL Record Number	**Record Type**	**SPINDL**	**300**	**CLW**
n	**2000**	**1031**	**300.0**	**60**

Figure 3-4. Type 2000 CL Record Structure

Figure 3-4 illustrates the following **SPINDL/300,CLW** statement:

ICLWRD (1) = Record number
ICLWRD (2) = 2000
ICLWRD (3) = 1031
CLWRD (4) = 300.0
ICLWRD (5) = 60

1	2	3	4	5	6	7	8	9
CL Record Number	**Record Type**	**PARTNO**						
n	**2000**	**1045**	**T H**	**I S**	***b*I**	**S*b***	**T E**	**S T**

Figure 3-5. Text in CL Record

The **PARTNO, PPRINT**, and **INSERT** statements have special Type 2000 formats, as shown below.

ICLWRD (1) = CL record number
ICLWRD (2) = 2000
ICLWRD (3) = 1044 or 1045 or 1046
ICLWRD (4-36) will contain the PARTNO, PPRINT or INSERT record.

Figure3-5 illustrates the handling of text in a CL record. Each text location contains two characters. A space is considered to be one character; (we show a space as $_b$ in the example).

1	2	3	4	5
CL Record Number	**Record Type**	**MACHIN**		**Machine Number**
n	**2000**	**1015**		***n***

Figure 3-6. MACHIN Statement CL Record Structure

The **MACHIN** statement has a unique Type 2000 format, as shown below:

ICLWRD (1) = CL record number
ICLWRD (2) = 2000
ICLWRD (3) = 1015
ICLWRD (4) unused
CLWRD (5) = machine number

The remainder of the parameters on the **MACHIN** statement are stored in the ICLWRD and CLWRD arrays in the same way as standard Type 2000 records.

3.4 Type 3000 Surface Definition Record

The canonical form of a circle or cylinder is stored in this type of record.

ICLWRD (1) = CL record number
ICLWRD (2) = 3000
ICLWRD (3) = 2 (code for drive surface)
ICLWRD (4) = 0
ICLWRD (5) = 4 for circle
5 for cylinder
ICLWRD (6) = unused
ICLWRD (7) = unused
ICLWRD (8) = unused
CLWRD (9) = x-coordinate of the center of the circle
CLWRD (10) = y-coordinate of the center of the circle
CLWRD (11) = z-coordinate of the center of the circle
CLWRD (12) = x-component of the axis of the surface
CLWRD (13) = y-component of the axis of the surface
CLWRD (14) = z-component of the axis of the surface
CLWRD (15) = radius of the circle

Immediately following a Type 3000 record will be a Type 5000 record that specifies motion along the defined surface.

3.5 Type 5000 Motion Record

All FIL motion statements, such as **GOTO**, and G**ODLTA** generate Type 5000 records. **GODLTA** is translated to **GOTO** in FIL.

ICLWRD (1) = CL record number
ICLWRD (2) = 5000
ICLWRD (3) = 3 for FROM
5 for GOTO, GO, etc.
6 for continuation record type
ICLWRD (4) = unused
ICLWRD (5) = unused
CLWRD (6) = x-coordinate of the 1st CL point
CLWRD (7) = y-coordinate of the 1st CL point
CLWRD (8) = z-coordinate of the 1st CL point
CLWRD (9) = x-coordinate of the 2nd CL point
CLWRD (10) = y-coordinate of the 2nd CL point
CLWRD (11) = z-coordinate of the 2nd CL point
.
.
.
CLWRD ((n*3)+3) = x coordinate of the nth CL point
CLWRD ((n*3)+4) = y coordinate of the nth CL point
CLWRD ((n*3)+5) = z coordinate of the nth CL point

With **MULTAX/OFF**, you can have a maximum of 80 CL points in a record. With **MULTAX/ON**, you can have a maximum of 40 CL points. A motion statement containing more than 80 CL points (or 40 CL points with **MULTAX/ON**) generates a continuation record. Continuation records immediately follow the primary Type 5000 record and are not preceded by a Type 1000 record. The following example illustrates the handling of a continuation record.

ICLWRD (1) = CL record number
ICLWRD (2) = 5000
ICLWRD (3) = 5
ICLWRD (4) = 0 (unused)
ICLWRD (5) = 0 (unused)
CLWRD (6) = 10.0 (1st CL point X)
CLWRD (7) = 10.0 (1st CL point Y)
CLWRD (8) = 10.0 (1st CL point X)
.
. (80th CL point)
ICLWRD (1) = CL record number
ICLWRD (2) = 5000
CLWRD (3) = 6 (to signify a continuation record)
ICLWRD (4) = 0 (unused)
ICLWRD (5) = 0 (unused)
CLWRD (6) = 10.0 (81st CL point X)
CLWRD (7) = 10.0 (81st CL point Y)
CLWRD (8) = 10.0 (81st CL point Z)
.
. (end of record)

1	2	3	4	5	6	7	8
CL Record Number	**Record Type**	**GOTO**					
n	**5000**	**5**			**10.0**	**12.0**	**15.0**

Figure 3-7. Type 5000 Motion CL Record Structure

Figure 3-7 illustrates the following **GOTO/10,12,15** CL Record.

ICLWRD (1) = CL record number
ICLWRD (2) = 5000
ICLWRD (3) = 5
ICLWRD (4) = 0
ICLWRD (5) = 0
CLWRD (6) = 10.0
CLWRD (7) = 12.0
CLWRD (8) = 15.0

The following example illustrates the structure of a **MULTAX** record:

ICLWRD (1) = CL record number
ICLWRD (2) = 5000
ICLWRD (3) = 3 for FROM
5 for GOTO, GO, etc.
6 for continuation record type
ICLWRD (4) = unused
ICLWRD (5) = unused
CLWRD (6) = x-coordinate of the 1st CL point
CLWRD (7) = y-coordinate of the 1st CL point
CLWRD (8) = z-coordinate of the 1st CL point
CLWRD (9) = i-coordinate of the 1st CL point
CLWRD (10) = j-coordinate of the 1st CL point
CLWRD (11) = k-coordinate of the 1st CL point
.
. (40th CL point)
ICLWRD (1) = CL record number
ICLWRD (2) = 5000
ICLWRD (3) = 6 for continuation record type
ICLWRD (4) = unused
ICLWRD (5) = unused
CLWRD (6) = x coordinate of the 41st CL point X
CLWRD (7) = y coordinate of the 41st CL point Y
CLWRD (8) = z coordinate of the 41st CL point Z
CLWRD (9) = i coordinate of the 41st CL point I
CLWRD (10) = j coordinate of the 41st CL point J
CLWRD (11) = k coordinate of the 41st CL point K

. (end of record)

3.6 Type 6000 Records

CUTTER, **INTOL**, and **OUTTOL** statements have the same format as standard Type 2000 records.

CUTTER is a ***Type* 6000, *subtype* 6** CL Record
INTOL is a ***Type* 6000, *subtype* 4** CL Record
OUTTOL is a ***Type* 6000, *subtype* 5** CL Record

1	2	3	4	5
CL Record Number	**Record Type**	**Record Subtype**	**.500**	**.250**
n	**6000**	**6**	**.500**	**.250**

Figure 3-8. **Type 6000 CUTTER CL Record Structure**

Figure 3-8 illustrates the following CUTTER/.5,.25 statement:

ICLWRD (1) = Record number
ICLWRD (2) = 6000
ICLWRD (3) = 6
CLWRD (4) = .500 Cutter Diameter
CLWRD (5) = .250 Corner Radius

3.7 Type 9000 MULTAX Record

This record indicates whether subsequent motion records will contain tool axis vector components.

1	2	3	4
CL Record Number	**Record Type**	**MULTAX**	**ON**
n	**9000**	**2**	**1**

Figure 3-9. **Type 9000 MULTAX CL Record Structure**

Figure 3-9 illustrates the following **MULTAX/ON** CL Record.

ICLWRD (1) = CL record number
ICLWRD (2) = 9000
ICLWRD (3) = 2 (Code for **MULTAX**)
ICLWRD (4) = 1 for **MULTAX** or **MULTAX/ON**
0 for **MULTAX/OFF**

Motion records generated while **MULTAX/ON** is in effect will contain the IJK tool axis vector components.

3.8 Type 14000 FINI Record

This is the last logical record in the CL file. It corresponds to the **FINI** statement in the ASCII CL file.

1	2	3
CL Record Number	**Record Type**	**FINI**
n	**14000**	**0**

Figure 3-10. **Type 14000 FINI CL Record Structure**

Figure 3-10 illustrates the **FINI** CL Record.

ICLWRD (1) = CL record number
ICLWRD (2) = 14000
ICLWRD (3) = 0

4 Command Language

Introduction

Austin N.C., Inc.'s Factory Interface Language (FIL) command language is a very robust and powerful set of programming tools allowing you to enhance the post processors capability. This chapter explains the file and command formats, the computing capabilities, the logic capabilities, the text handling capabilities, the geometric definitions and all other FIL functions.

4.1 File and Command Format

A FIL file consist primarily of two types of sections, FIL subroutines are considered one section, and anything not enclosed in a FIL subroutine is considered as being in the Global Area. Anything defined in the Global Area is initialized at the beginning of the post processor execution and is available for use by any of the FIL subroutines. A FIL subroutine starts with the **CIMFIL/ON,*Type,Subtype*** command and ends with the **CIMFIL/OFF** command. You are allowed as many FIL subroutines as necessary as long as each FIL subroutine is unique. In other words you cannot have two FIL routines for the same CL record. An example of a FIL subroutine follows:

```
.
X=10
CIMFIL/ON,LOADTL
    DMY = POSTF(21)
    DMY = POSTF(13)
CIMFIL/OFF
Y=10
.
```

In the above example the variables X and Y are defined outside the FIL subroutine and are global to any FIL subroutine. The commands within the **CIMFIL/ON** and **CIMFIL/OFF** boundaries are only executed when a **LOADTL** CL record is read from the CL file.

4.1.1 Input Formats

The FIL file processor reads only the first 72 columns in a line, any data entered beyond column 72 is ignored by FIL.

Within the first 72 columns, you may write statements in whichever columns you desire. FIL ignores blank spaces, so you may include them where you wish in order to make the file easier to read. For example, the following two statements, the first with no embedded blank and the second with embedded blanks, are equivalent:

```
TLNUM=POSTF(7,4)
 TLNUM   =    POSTF(  7 , 4 )
```

The FIL language is **NOT** case sensitive, uppercase and lowercase characters are all converted to uppercase prior to FIL processing. Therefore the following commands are equivalent:

```
TLN = POSTF(7,4)
Tln=postf(7,4)
tln=postf(7,4)
TLN=Postf(7,4)
```

Normally, one statement is included in each line, but it is possible for a single statement to be continued on one or more successive lines and conversely, for several short statements to be included in one line. Furthermore, comments may be included in a line along with statement information. These capabilities are provided by the single dollar sign, double dollar sign, and semicolon.

4.1.1.1 The Single Dollar Sign

The single dollar sign indicates the end of statement information in the current line and further indicates that the current statement is continued on the next line. A statement can be continued on any number lines as long as each one but the last one contains a single dollar sign. Any information to the right of the single dollar sign is ignored and is not considered as part of the statement, but does appear in a printout of the file, so you may place comments there if you desire.

For example, the following statement is split into three lines. The first contains a comment to the right of the dollar sign

```
M1=MACRO/          $      THIS IS A MACRO
          TL=1,    $
          OFN=0
.
.
TERMAC
```

4.1.1.2 The Double Dollar Sign

The double dollar sign indicates the end of the current statement prior to column 72. Comments can be placed to the right of the double dollar sign. If no statement appears to the left of the double dollar sign, the line serves solely as a comment line and has no effect on the FIL processor.

Example:

```
$$ This is a comment line
M1=MACRO/TL=1,OFN=0      $$ THIS IS A MACRO
.
.
TERMAC                   $$ END OF MACRO
```

4.1.1.3 The Semicolon (;)

The semicolon terminates the current statement in such a way that the next statement starts immediately following the semicolon in the same line. Thus, you can use the semicolon in order to include several short statements in a single line.

Example:

```
A=1;B=2;C=3;D=4;E=5
```

The above line contains five statements and is equivalent to the following five lines.

```
A=1
B=2
C=3
D=4
E=5
```

Some FIL statements consist of a single word. A comma following such a statement is treated like a semicolon, indicating the end of one statement and the beginning on the next.

The following examples illustrate input in which the comma and semicolon are equivalent:

```
RAPID,GOTO/0,0,0
RAPID;GOTO/0,0,0

GOHOME,END,STOP
GOHOME;END;STOP
```

4.1.2 Statements and Their Elements

The statement is the basic unit of input within a FIL file. It is comparable to a sentence in the English language, expressing at least one complete instruction or unit of information.

A statement is composed of vocabulary words, numbers, symbols, and special characters used for punctuation or as operators. For example, consider the following statement:

```
SPINDL / 500 , RPM , CLW , RANGE , RNG
```

In this statement, **RNG** is a symbol, **SPINDL, RPM, CLW,** and **RANGE** are vocabulary words, **500** is a number, and the slash and commas are punctuation characters.

These elements and types are described in the following sections.

4.1.3 Numbers

Standard mathematical notation is used to represent numbers in the FIL language. If a number has a fractional part, a decimal point is used to separate the integer part from the fractional part. If a number has no fractional part, a decimal point is not required but is permissible.

A minus sign preceding a number indicates that it is negative. A plus sign may precede a positive number but in unnecessary since a number not preceded by either a plus on minus sign is assumed to be positive.

The following are examples of numbers as they can be expressed in the FIL language:

1 5 6.625 -7.75 -4 -103.9878

The following are equivalent representations of the same number:

2 2. 2.0 +2 +2.0 +2.0000

4.1.4 Vocabulary Words

A vocabulary word is a word that has built in meaning to FIL. Vocabulary words consist of six or fewer characters. Most words consist entirely of alphabetic characters, though a few also contain numeric characters. Some vocabulary words are obviously based on more than on English word but are still considered single words by FIL. For example, the FIL word **LOADTL** is based on the English words “load” and “tool”.

There are two types of vocabulary word - major and minor.

Major Words

A major word is so called because it is the most important word in the statement, being the word that establishes the basic meaning of the statement. Some major words express a complete meaning by themselves and can stand alone; for example, **RAPID** and **STOP**.

Other major words require additional information. In these cases, the major word is written to the left of a slash and the additional information to the right.

Example:

```
LOADTL/1,ADJUST,1
SPINDL/500,RPM
```

Minor Words

A minor word is one that can only appear to the right of the slash.

Example:

```
LOADTL/1,ADJUST,1,LENGTH,0
SPINDL/500,RPM
```

ADJUST, **LENGTH**, and **RPM** are minor words. The minor entries in a statement are separated by commas.

4.1.5 Symbols

A symbol is a name that you create and assign to something that you are defining. You may subsequently reference what you have defined by its symbol. For example, you could write the following statement:

P1 = POINT / 0 , 5 , 0

This statement assigns the symbol **P1** to the point with the specified coordinates. Later in the process you could refer to this point by its symbol in a statement such as:

GOTO / P1

A symbol can consist of from one to six alphabetic and/or numeric characters, at least one of which must be alphabetic.

If a vocabulary word is used as a symbol, it is disabled as a vocabulary word for the remainder of the FIL process and is treated as a symbol wherever it appears subsequently.

You will find it helpful to use symbols that are indicative of what they represent. For example, **P1, P2**,etc are meaningful symbols for points; **TLN** for a tool number; **NCH** for number of characters.

4.2 Statement Labels

You can attach a label to a statement and refer to the statement from other statements by using the label. (Statement types that refer to other statements include **IF** and **JUMPTO**). Like a regular symbol, a label can consist of from one to six alphabetic and/or numeric characters, but unlike a regular symbol, all the characters can be numeric.

A label must be the first element in its statement and must be followed by the right parenthesis, which separates the label from the rest of the statement but is not part of the label itself. You can include a label on any statement, but you should do so only on statements that you intend to refer to from other statements.

A label does not necessarily have to be unique throughout an entire FIL file. The same label can be used in different loops and macros as well as once in the main subroutine section outside any loops or macros. This is significant because it means that you can use standard macros in a series of FIL subroutines without being concerned about duplicate labels.

Example:

```
S1)     GOTO/1,1,0
A)      FEDRAT/01,IPM
456)    B = 10 / 2 * (3+4)
```

In the above statements, **S1, A**, and **456** are statement labels.

4.3 SYN (Synonym)

The **SYN** (synonym) statement permits you to create your own alternate spellings of FIL vocabulary words.

Its format is as follows:

SYN/*new spelling, vocab, new spelling, vocab, etc....*

a variable number of pairs of entries may be specified. The first entry of each pair is the new spelling for the vocabulary word that is the second entry of the pair.

Example:

SYN/CM,CIMFIL,MC,MACRO,TX,TEXT

This statement establishes **CM** as an alternate spelling for **CIMFIL**, **MC** for **MACRO**,and **TX** for **TEXT**. Once these equivalencies have been established, either the original spellings or the new ones can be used interchangeably.

Synonyms can be specified for vocabulary words only, not for symbols.

If a specified synonym is itself a vocabulary word, then the word loses its original meaning and takes on the meaning of the word to which it is equivalence.

Example:

SYN/AT,ATANGL

AT is normally a vocabulary word, but this statement causes **AT** to be treated as if it were **ATANGL**.

A set of standard synonyms are built into the FIL command language and can be activated with the following command:

SYN/ON

It is permissible to activate the standard synonyms and specify additional synonyms with the regular **SYN** statement.

The following is a list of the standard synonyms and their equivalent vocabulary words.

AA	ATANGL	DI	DIAMTR	LR	LINEAR	RT	RIGHT
AV	AVOID	DR	DRILL	OB	OBTAIN	RO	ROTABL
BO	BORE	FC	FACE	OS	OPSTOP	SP	SPINDL
CE	CENTER	FD	FEDRAT	OR	ORIGIN	TT	TANTO
CK	CHECK	GD	GODLTA	LL	PARLEL	TH	THREAD
CP	CLEARP	GT	GOTO	PP	PERPTO	TU	TURN
CO	COOLNT	IO	INTOF	PT	POINT	TE	TURRET
CC	CUTCOM	IP	INDIRP	RA	RADIUS	VE	VECTOR
CU	CUTTER	IV	INDIRV	RP	RAPID		
CY	CYCLE	LF	LEFT	RE	RETRCT		

4.4 ALIAS

ALIAS/%nam1,nam2

The **ALIAS** command can be used in **FIL** files to make the variable names and syntax to be more readable.

%nam1: is the aliased name which should be replaced by its equivalence name nam2.

nam2: is any FIL command, use $ continue to the next line.

Notes:

1. **nam1** must start with %
2. **nam2** cannot contain %
3. **nam1** and **nam2** will be automatically upper-cased
4. **nam2** must be valid FIL commands
5. **nam1** cannot exceed 60 chars
6. **nam2** cannot exceed 512 chars, use **CALL/MACRO** instead
7. **$** in **nam2** will continue onto next line
8. **$$** in **nam2** will end the current line
9. no blanks are allowed in Alias command
10. a comma must be used to separate **nam1,nam2**
11. **PRINT/ON,ALIAS** will print the translation during Post execution as a debug aid.

Possible errors generated:

Error BAD ALIAS/cmd MUST START WITH %
Error BAD ALIAS/cmd MISSING COMMA SEPERATOR
Error BAD ALIAS/cmd BLANK USED AS SEPERATOR
Error TOO MANY ALIAS/commands EXCEEDS 3000
Error BAD ALIAS/cmd ALREADY USED
Error BAD ALIAS/cmd LINE TOO LONG
Error BAD ALIAS/cmd 1ST NAME EXCEEDS 60 CHARS
Error BAD ALIAS/cmd 2ND NAME EXCEEDS 512 CHARS
Error BAD ALIAS/cmd 1ST NAME HAS NON-ALPHA CHAR
Error BAD ALIAS/CMD 2ND NAME HAS % CHAR

Example:

The following is a sample example of using **ALIAS** for capturing the **ROTABL** command and output G90 first.

Define ALIAS commands:

```
ALIAS/%Rotabl_Begin,cimfil/on,ROTABL
ALIAS/%Process_CL_Record,dmy=postf(13)
ALIAS/%Save_CL_Record,dmy=postf(20)
ALIAS/%Restore_Saved_CL_Record,dmy=postf(21)
ALIAS/%G,7
ALIAS/%Output,POSTN/OUT,
ALIAS/%End,cimfil/off
```

Use ALIAS commands:

```
%Rotabl_Begin
%Same_CL_Record
%Output,%G,90
%Restore_Saved_CL_Record
%Process_CL_Record
%End
```

Same as:

```
CIMFIL/ON,ROTABL
      DMY=POSTF(20)
      POSTN/OUT,7,90
      DMY=POSTF(21)
      DMY=POSTF(13)
CIMFIL/OFF
```

*****Special ALIAS command usage for Unigraphics or other systems that produce vocabulary words longer than 6 characters:**

Unigraphics NXx version can now output "major or minor" words longer than 6-characters (i.e. **SETAXIS/W5.000**) in the CLS file. To support this the **ALIAS/a>b** command has been added to the **UG-interface**.

ALIAS/a>b can be added to the **uncas1.vtb** or into the CLS file directly. The interface will apply a simple substitution for the long word, string **a**, to the short word, string **b**, and then pass it to the ACL or APT file as a G-Post command..

Example:

CLS File-

```
$$ CLS file has long word SETAXIS:
PARTNO TEST ALIAS IN CLS
ALIAS/SETAXIS>SET
ALIAS//W>TABLE,
SPINDL/300
FEDRAT/10
LOADTL/1
GOTO/1,0,0
SETAXIS/W5.000
GOTO/2,0,0
END
FINI
```

ACL File-

```
$$ ACL file contains the converted SETAXIS as SET and the converted /W as /TABLE,.
$$ The value ”5.000” would be output after the comma.
PARTNO TEST ALIAS IN CLS
INCLUD/UNCAS8.VTB
REMARK ALIAS/SETAXIS>SET
REMARK ALIAS/W5>TABLE,5
SPINDL/300
FEDRAT/10
LOADTL/1
GOTO/1,0,0
SET/TABLE,5.000
GOTO/2,0,0
END
FINI
```

FIL Example:

$$ FIL file generates G0 Wn for SETAXIS/cmd, remember it is being converted to SET/TABLE,n.

```
REDEF/ON
$$ PROCESS SET/TABLE VIA ALIAS/CMD
CIMFIL/ON,SET
IW4=POSTF(7,4)
IW5=POSTF(7,5)
CASE/IW4
     WHEN/(ICODEF(TABLE))
       POSTN/OUT,7,0,23,IW5              $$ OUTPUT G0 Wn
     WHEN/OTHERS
       DMY=POSTF(13)
ENDCAS
CIMFIL/OFF

FINI
```

4.5 The INCLUD Statement

INCLUD/*Filename*

This statement is used to include the contents of the specified ASCII source file, ***Filename,*** into the current file. Once the contents of ***Filename*** are included the process will continue on the line following the INCLUD statement.

Filename can be any valid ASCII text file on your system. If the file extension is not specified, **.INC** is used. If the ***Filename*** does not include the path the system will locate the file as follows:

1. Search the current working directory where the source file is located.

2. Search the directory specified by the **UNC$INCLUDE** path from the **Config.tbl** file. It is recommended that the **UNC$INCLUDE** and the **UNC$LIBRARY** paths be the same in the **Config.tbl** file.

3. Search the directory specified by the **UNC$LIBRARY** from the **Config.tbl** file.

- If the specified ***Filename*** is not located in one of these locations the system will generate an error message.

- The **INCLUD/*Filename*** statement must start in column one on any line of your source file. Shift this command to the right out of column one and the system will generate an error message.

- Comments are not allowed in the **INCLUD/*Filename*** statement.

- Four levels of **INCLUD** are permitted.

Examples:

The following **INCLUD** statements are **INVALID**:

```
$$ No comments allowed on the INCLUD line
INCLUD/TEST1.INC          $$ THIS COMMENT WILL CAUSE AN ERROR!

$$ INCLUD statement must start in column one
  INCLUD/TEST2.INC
```

The following **INCLUD** statements are **VALID**:

```
$$ Including TEST1.DAT, looks through the defined search paths
INCLUD/TEST1.DAT

$$ Exact location of file is specified
INCLUD/C:\TEST\SOURCE\MACHINE1\TEST1.INC

$$ Including TEST3.INC, looks through the defined search paths
INCLUD/TEST3
```

4.6 The INCLUD/BINARY Statement

INCLUD/BINARY,*Filename*

This statement is used to include the contents of the specified encrypted binary file, ***Filename,*** into the current FIL file. Once the contents of ***Filename*** are included the process will continue on the line following the INCLUD/BINARY statement.

The ***Filename*** is an encrypted binary file created by the Austin N.C. utility **Wncrypt.exe**. The purpose is to protect or lock the FIL source data from modification.

This encryption method is a lot simpler than the system macros method of **PUNCH-READ/20** commands. The encrypted file can be any FIL text unlike the system macros. You can continue to use both methods.

Caution: Once a FIL file is encrypted, it cannot be decrypted. So save your original source FIL file.

On Windows, you can encrypt a file by running **\CAMSYS\Wncrypt.exe**.
On UNIX, you can encrypt a file by running **/unc/camsys/ncrypt**.

Wncrypt.exe or **ncrypt** will prompt for the input and output file names or you can pass them as arguments as **"wncrypt test1.dat test1.bin"**, to encrypt the source file **test1.dat** into **test1.bin**. We suggest you name the output files as *.bin so you can identify them as encrypted file.

When FIL encounters a **INCLUD/BINARY,*Filename*** command, it will search the local directory and then the **UNC$LIBRARY** for the specified ***Filename***.

Filename is any legal name and is not restricted to 6-characters.

INCLUD/BINARY,*Filename* must start in column-1 and contain no blanks.

There is no limit on the number of **INCLUD/BINARY,*Filename*** commands allowed in FIL.

INCLUD/BINARY,*Filename* can be anywhere in the FIL file since it is treated like any other FIL input except it has been encrypted.

Example:

1. Create an encrypted file, **cm01.dat,** of the machin section for the post number one.

Using a text editor, create the file to be encrypted, ***cm01.dat***:

```
CIMFIL/ON,MACHIN
   MCH=POSTF(7,5)
   DMY=POSTF(13)
CIMFIL/OFF
```

Run the encryption routine from the command prompt or windows explorer.

Windows: > ***wncrypt cm01.dat cm01.bin***
UNIX: > ***ncrypt cm01.dat cm01.bin***

Copy the encrypted file to the CAMLIB directory

Windows: > ***copy cm01.bin \camlib***
UNIX: > ***cp cm01.bin /unc/camlib/cm01.bin***

2. Using the encrypted file in **uncx01.f01** FIL file.

File uncx01.f01:

```
CIMFIL/ON,PARTNO
  TPN=TEXT/CLW
  DMY=POSTF(13)
CIMFIL/OFF

INCLUD/BINARY,CM01.BIN

CIMFIL/ON,SPINDL
   DMY=POSTF(13)
CIMFIL/OFF
FINI
```

4.7 Computing

In many FIL statements that you program, you need to specify numerical values. Often, you will not know these values directly but must compute them. The FIL language allows you to write computing expressions that will cause the computer to perform these computations for you.

4.7.1 Scalar Assignment

Computing expressions specify relationships involving scalars. A scalar is defined to be either a literal number or a symbol that represents a number. A symbol can be assigned to a number the following way:

Symbol = Number

Once you have defined a scalar symbol, you can use it wherever a literal number could be used. Using a symbol instead of a literal number can help to make FIL programming more general and easier to modify.

For example, suppose you need to write a particular dimension, 2.5 in ten statements of the FIL subroutine. If it later becomes necessary to change the dimension to 2.75, you will have to modify ten statements. As an alternative, you could assign a symbol to the dimension by means of a statement like the following:

D1 = 2.5

Then, in each statement where the dimension is required, you write **D1** instead of 2.5. To change the dimension when the program is written this way, you need only change one statement:

D1 = 2.75

4.7.2 Arithmetic Operators

You can write computing expressions that cause arithmetic operations to be performed by using the following arithmetic operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The double asterisk (**), though physically two characters, is treated logically as a single operator.

Any number of scalar symbols, literal numbers, and arithmetic operators can be combined into computing expressions of any desired level of complexity.

When several operators appear in an expression, you can use parentheses to indicate the order in which they are to be performed. In the absence of parentheses, operators are performed in the following order:

1. Exponentiation (**)
2. Multiplication and division (* and /)
3. Addition and subtraction (+ and -)

Operators at the same level (* and / , + and -) are executed from left to right. Where there is more than one level of parentheses, the innermost levels are resolved first.

Example:

```
A = 2*6 + 5
```

Since multiplication is performed before addition, this expression is evaluated as if it were written:

```
A = (2*6) + 5
```

The result, therefore, is that **A** is set equal to **17.**

Parentheses could be used to cause the addition to be performed before the multiplication:

```
A = 2*(6 + 5)
```

The result, therefore, is that **A** is set equal to **22.**

The following example illustrates how several levels of parentheses are handled:

```
X = (4 * ( (2+1)**2 - 1) ) / 8
X = (4 * ( 3**2 - 1) ) / 8
X = (4 * ( 9 - 1) ) / 8
X = (4 * 8 ) / 8
X =  32 / 8
X =  4
```

It is permissible to write two consecutive operators as follows:

```
B = 2 * -1
```

A scalar symbol can be redefined, even in terms of itself.

Example:

```
N = 1
.
.
N = N + 1
```

This statement cause FIL to take the current value of **N**, which is 1, add 1 to it, which produces 2, then store 2 as the new value of **N.**

The FIL language permits you to write computing expressions not only in statements by themselves but also in other types of statements where a literal number or scalar symbol would normally appear. The expression is enclosed within parentheses and is called a nested computation.

Example:

```
X = 5
N = 4 + X
TMP = POSTF(7,N)
```

These three statements can be combined into one, as follows:

```
TMP = POSTF(7, (N = 4 + X) )
```

If it is not necessary to refer to the result of the computation subsequently, then you do not have to assign a symbol to it:

```
TMP = POSTF(7, (4 + X) )
```

4.7.3 Computation Functions

The FIL language has several built-in procedures called functions that perform particular operations on designated scalars or geometric entities called arguments of the function, which are written in parentheses following the function name. The result of the execution of the function is always a scalar; therefore, a function reference can be written in a computing expression wherever a scalar can be written.

For example, the word **SINF** activates the procedure that determines the sine of whatever angle is written in the parentheses following **SINF**. Thus, the function reference **SINF(30)** causes the sine of 30 degrees to be determined, which is .5. The following example illustrates various expressions using **SINF**:

```
ANG = 30
A = SINF(ANG)          ( A = .5)
B = 1 + SINF(ANG)      ( B = 1.5)
C = 2 * SINF(30)       ( C = 1)
```

Following are descriptions of all the FIL functions in alphabetical order.

4.7.3.1 The ABSF(scalar) Function

The result is the absolute value of the scalar.

```
A = -30
B = ABSF(A)            ( B = 30)
```

4.7.3.2 The ACOSF(scalar) Function

The result is the angle that has the specified cosine.

```
A = .70711
B = ACOSF(A)           ( B = 45)
```

4.7.3.3 The ANGLF(point 1,point 2) Function

The result is the angle measured from the positive x-axis to a unit vector, defined as a vector from point 1 to point 2, measured in the counter clockwise direction in the xy-plane.

```
A =  POINT/ 0,  0,  0
B =  POINT/ 1,  1,  0
C =  ANGLF(A,B)                (C = 45)
```

4.7.3.4 The ANGLF(vector) Function

The result is the angle measured from the positive x-axis to the unit vector measured in the counter clockwise direction in the xy-plane.

```
A =  VECTOR/ 1 , 1 , 0
B =  ANGLF(A)          (B = 45)
```

4.7.3.5 The ASINF(scalar) Function

The result is the angle that has the specified sine.

```
A = .7070
B = ASINF(A)           ( B = 45)
```

4.7.3.6 The ATANF(tangent) Function

The result is the angle that has the specified tangent. The angle is in the first quadrant if the tangent is positive and in the fourth quadrant if the tangent in negative.

```
A = 1
B = ATANF(A)           ( B = 45)
A = -1
B = ATANF(A)           ( B = -45)
```

4.7.3.7 The ATAN2F(y,x) Function

The result is the angle whose tangent is y/x. Unlike the single argument arc tangent function **ATANF,** which can only produce an angle in the first or fourth quadrant, this function can produce an angle in any quadrant, based on the signs of the two arguments.

```
A = ATAN2F(0,1)        (A=0)
A = ATAN2F(1,0)        (A=90)
A = ATAN2F(0,-1)       (A=180)
A = ATAN2F(-1,0)       (A=-90)
A = ATAN2F(1,1)        (A=45)
A = ATAN2F(1,-1)       (A=135)
A = ATAN2F(-1,1)       (A=-45)
A = ATAN2F(-1,-1)      (A=-135)
```

4.7.3.8 The CANF(symbol,scalar) Function

The result is the element designated by the scalar from the canonical form designated by the symbol. A scalar equal to 1 selects the first element; 2, the second, etc.

```
P1 = POINT / 13 , 24 , 3 5
XVAL = CANF(P1,1)      (XVAL = 13)
YVAL = CANF(P1,2)      (YVAL = 24)
ZVAL = CANF(P1,3)      (ZVAL = 35)
```

4.7.3.9 The COSF(scalar) Function

The result is the cosine of the specified angle.

```
A = -30
B = COSF(A)          ( B = .86603)
```

4.7.3.10 The DISTF(point,point) Function

The result is the distance between the two specified points, the order of the points is irrelevant; the distance is always positive.

```
P1 = POINT / 1 , 4 , 0
P2 = POINT / 5 , 7 , 0
D1= DISTF(P1,P2)          ( D1 = 5)
```

4.7.3.11 The DOTF(vector,vector) Function

The result is the dot product of the two vectors. If vector **A** has components *ax, ay , az,* and vector **B** has components *bx, by, bz,* then **A** dot **B** equals:

*ax * bx + ay * by + az * bz*

```
A = VECTOR / 2 , 3 , 0
B = VECTOR / 4 , 1 , 0
D = DOTF(A,B)          ( D = 11)
```

4.7.3.12 The EXPF(scalar) Function

The result is the value of *e* (the base for natural logarithms) raised to the power specified by the scalar.

```
A = EXPF(.69315)          (A = 2)
```

4.7.3.13 The INTF(scalar) Function

The result is the specified scalar with its fractional part, if any, removed.

```
A = INTF(2)          (A = 2)
A = INTF(2.1)        (A = 2)
A = INTF(2.9999)     (A = 2)
A = INTF(-7.9989)    (A = -7)
A = INTF(123.9897)   (A = 123)
```

4.7.3.14 The LNTHF(vector) Function

The result is the length of the specified vector.

```
V1 = VECTOR / 3 , 0 , 0
A = LNTHF(V1)        (A = 3)
```

4.7.3.15 The LOGF(scalar) Function

The result is the natural logarithm (base *e*) of the scalar.

```
A = LOGF(2)          (A=.69315)
```

4.7.3.16 The MAXF(scalar 1, scalar 2, ---, scalar n) Function

The result is the scalar with the maximum value, A variable number of scalars may be specified.

```
A = 4.7
B = -5.6
C = 7.6
D = -10.5
G = MAXF(A,B,C,D)    (G = 7.6)
```

4.7.3.17 The MINF(scalar 1, scalar 2, ---, scalar n) Function

The result is the scalar with the minimum value, A variable number of scalars may be specified.

```
A = 4.7
B = -5.6
C = 7.6
D = -10.5
G = MINF(A,B,C,D)    (G = -10.5)
```

4.7.3.18 The MODF(scalar 1, scalar 2) Function

The result is scalar 1 modulo scalar 2; that is it is the remainder when scalar 1 is divided by scalar 2.

```
A = 123.5
B = MODF(A,1)        (B = .5)
C = MODF(A,10)       (C = 3.5)
D = MODF(A,100)      (D = 23.5)
```

4.7.3.19 The SIGNF(scalar 1, scalar 2) Function

The result is a scalar with the numerical value of scalar 1 and the sign of scalar 2.

```
A = 123.5
B = -16.5
C = SIGNF(A,B)       (C = -123.5)
D = SIGNF(B,A)       (C =  16.5)
```

4.7.3.20 The SINF(angle) Function

The result is the sine of the specified angle.

```
A = 30
B =  SINF(A)         (B = .5)
```

4.7.3.21 The SQRTF(scalar) Function

The result is the square root of the scalar.

```
A = 100
B =  SQRTF(A)          (B = 10)
```

4.7.3.22 The TANF(angle) Function

The result is the tangent of the specified angle.

```
A =  0
B =  TANF(A)           (B = 0)
A =  30
B =  TANF(A)           (B = .57735)
A =  45
B =  TANF(A)           (B = 1)
```

4.8 The PPWORD Statement

PPWORD/*word,IC*

PPWORD enables you to add post processor words to the FIL system.

word is the post word and ***IC*** is the integer code.

These are the legal integer codes:

- 3101 to 4095 Major or Minor word. Using any other values can corrupt the system defined Major and Minor words and cause undesirable results.

Caution:	**If you use any other *IC* number assignment, you will probably get a runtime error.** **We strongly suggest that you appoint a site coordinator to administer your *word,IC* additions. Without that control, you run the risk of creating some enormous problems.**

You can use an integer code that's already in use, thereby changing the assignment.

You must add the **PPWORD/*word,IC*** statement to each part program and each FIL file that uses the new word/integer code assignment.

4.9 The PRINT Statements

PRINT/ON-OFF

The **PRINT/ON** statement causes a printout to be produced of each named scalar value and canonical form as it is generated.

The **PRINT/OFF** turns off a previous **PRINT/ON** statement and is the assumed setting if neither **PRINT/ON** nor **PRINT/OFF** is specified.

PRINT/ON is a very handy debugging tool within FIL, but be aware it generates a large amount of data in the Listing File (.LST).

PRINT/ON,IN-OFF,IN

The **PRINT/OFF,IN** statement turns off the printout of the FIL file in the LIST file.

The **PRINT/ON,IN** turns on a previous **PRINT/OFF,IN** statement and is the assumed setting if neither **PRINT/ON,IN** nor **PRINT/OFF,IN** is specified.

4.10 The PUNCH/30 Statement

PUNCH/30,<filename>,[ALL-s1,s2]

The **PUNCH/30** statement is used to save the data into a file named ***filename***. You would place this command in the **FINI** section of the 1st post **(CIMFIL/ON,14)** section. You can use the ALL command to save all variables or selectively give the symbol names to be saved.

The file will be saved (deleted if one already exists) in the UNC$CANON entry of the config.tbl file. Do not specify a file type or extension.

The opposite command is **READ/30,<filename>** and you would place this in the second post and at the **CIMFIL/ON,MACHIN** section.

Example:

FIL File uncx01.f01>

```
$$ SAMPLE SAVE N,S
PRINT/ON
CIMFIL/ON,14
    DMY=POSTF(20)
    SQN=POSTF(1,3,479)
    SPD=POSTF(1,3,(291+19))
    $$ YOU CAN USE PUNCH/30,FILE01,ALL $$ TO PUNCH ALL SYMBOLS
    PUNCH/30,FILE01,SQN,SPD
    DMY=POSTF(21)
    DMY=POSTF(13)
CIMFI/OFF
```

FIL File uncx01.f02>

```
$$ SAMPLE READ N,S
PRINT/ON
CIMFIL/ON,MACHIN
    DMY=POSTF(13)
    READ/30,FILE01
CIMFIL/OFF
```

4.11 The TIMLIM Statement (x86 PC platforms only).

TIMLIM/*t* may be used to detect loops in FIL. The value ***t*** is given in minutes of processing time. If the value ***t*** is less than one, the run time is unlimited, which is the default as before.

The time termination uses the system elapsed time and **NOT** the CPU time and thus the termination point (**ISN)** may vary for the FIL file.

4.12 Subscripted Variables

You can assign the same symbol to a series of entities and designate a particular member of the series by means of a subscript. A subscript is a scalar, numeric or symbolic, or a computing expression yielding a scalar value that is enclosed within parentheses following a symbol.

When a symbol is subscripted, the symbol and the subscript taken together constitute the total "name" of the entity being defined or referred to. The usefulness of subscripts arises from the fact that arithmetic operations cannot be performed on a symbol but they can be performed on a subscript. Thus, subscripts make it possible to vary "names" by means of computing expressions.

4.13 RESERV

Before you can use a subscript with a particular symbol, you must indicate to FIL in advance that subscripts are to be allowed for the symbol by specifying it in a **RESERV** (reserve) statement. The format of the **RESERV** statement is:

RESERV/*symbol, maxs, symbol, maxs, ----*

A variable number of pairs of entries may be included. The first entry of a pair is a symbol and the second is the maximum allowable subscript for the symbol.

Example.

RESERV/A, 5, L1, 3, C1, 2

This statement permits the following symbol-subscript combinations to be used subsequently:

A(1), A(2), A(3), A(4), A(5), L1(1), L1(2), L1(3), C1(1), C1(2)

An error results if a subscript greater than the specified maximum is applied to a symbol. For example, with the above **RESERV** statement in effect, an error would result if you tried to use **A(6)**.

On the other hand, each possible subscript value does not have to be used. You could, for example, specify 5 as the maximum subscript value for **A** but only define **A(l), A(2),** and **A(3).** (This would, however, unnecessarily waste computer storage.)

Once a symbol has been identified as being subscripted by appearing in a **RESERV** statement, it must be subscripted whenever it is used (except in **MACRO** and **CALL** statements, as explained in the section on macros).

Subscripted variables are particularly useful in loops. A scalar variable can be altered each time through the loop and then be used as a subscript. This variation in subscript value serves to provide a different name to each of the series of entities being referred to.

Only integer values can be used as subscripts. If a non-integer value is specified, it is truncated to the next lower integer.

As an example, consider the following statements. Five points along the x-axis are defined with the distance between adjacent points being 1.5. The points are identified as **P(1), P(2), P(3), P(4)**, and **P(5).**

```
RESERV/P, 5
Y=1.5
N=1
ID1)     P(N)=POINT/N,0,0
Y=Y+1.5
N=N+1
IF(N-5)ID1, ID1, ID2
ID2)
```

Subsequently, whenever one of these points is referenced, a subscript must be specified to indicate which of the five points is referenced.

```
GOTO/P(3)
GOTO/P(1)
GOTO/P(4)
GOTO/P(2)
GOTO/P(5)
```

It is not necessary that all members of a subscripted array be the same type. For example, the following statements define the first member as a point, the second as a vector, and the third as a scalar:

```
RESERV/ARRAY, 3
ARRAY(1)  = POINT/0, 10, 1
ARRAY(2) = VECTOR/1,0,0
ARRAY(3) = 5
```

Macro names and statement labels cannot be subscripted.

4.14 Inclusive Subscripts

The inclusive subscript feature provides a simplified means of designating a series of members of a subscripted array in an FIL statement. The feature eliminates the need for writing the name of the subscripted array repeatedly, once for each desired member. Instead, the array name is written only once, followed by an inclusive subscript expression that designates the desired members of the array.

The general format of an inclusive subscript is:

(*a*, THRU, *b*, INCR, *c*)
DECR

a , ***b***, and ***c*** are non-zero scalars-numeric, symbolic, or nested computation. ***b*** may also be the word **ALL**. ***a*** and ***b*** must be positive.

- ***a*** specifies the lowest subscript.
- ***b*** specifies the highest subscript. If **ALL** is entered for ***b***, the maximum subscript value specified in the **RESERV** statement is used.
- ***c*** specifies the interval for stepping through the array.

If ***a*** or ***c*** is omitted, a value of 1 is assumed.

INCR for positive ***c*** specifies the order of stepping to be from ***a*** to ***b***, up by increments of ***c.***

DECR for positive ***c*** specifies the order of stepping to be from ***b*** to ***a***, down by increments of ***c***.

A negative ***c*** reverses the effect of **INCR** or **DECR**. **INCR**,***c*** is equivalent to **DECR,-*c*** and **DECR,*c*** is equivalent to **INCR,-*c*.**

Positive stepping starts at ***a*** and steps up to ***b*** or to the closest value not exceeding ***b*** if ***b*** cannot be reached exactly.

Similarly, negative stepping starts at ***b*** and steps down to the closest value greater than or equal to ***a***.

If the value of ***b*** is less than ***a***, a single value, ***a*** or ***b***, depending on the direction of stepping, is the result.

The general format of an inclusive subscript, as shown above, can contain up to five entries. However, not all entries need always be included. The following is a complete list of all allowable formats. The **RESERV** value is indicated by ***r***.

1.	**(ALL)**	**1 to *r* by 1**
2.	**(ALL, INCR)**	**Same as 1**
3.	**(ALL, DECR)**	***r* to 1 by -1**
4.	**(ALL, INCR, *c*)**	**1 to *r* by *c***
5.	**(ALL, DECR, *c*)**	***r* to 1 by *c***
6.	**(THRU, ALL)**	**Same as 1**
7.	**(THRU, ALL, INCR)**	**Same as 1**
8.	**(THRU, ALL, DECR)**	**Same as 3**
9.	**(THRU, ALL, INCR, *c*)**	**Same as 4**
10.	**(THRU, ALL, DECR, *c*)**	**Same as 5**
11.	**THRU, ALL)**	***a* to *r* by 1**
12.	**THRU, ALL, INCR)**	**Same as 11**
13.	**THRU, ALL, DECR)**	***r* to *a* by -1**
14.	**THRU, ALL, INCR, *c*)**	***a* to *r* by *c***
15.	**THRU, ALL,DECR, *c*)**	***r* to *a* by *c***
16.	**THRU, *b*)**	***a* to *b* by 1**
17.	**THRU, *b*,INCR)**	**Same as 16**
18.	**THRU, b,DECR**	***b* to *a* by -1**
19.	**THRU, b,INCR, *c*)**	***a* to *b* by *c***
20.	**THRU, b,DECR, *c*)**	***b* to *a* by *c***

Restriction

No more than 10 inclusive subscript expressions may be included in a single statement. This limit applies to the number of expressions, not to the number of subscripts generated per expression, which is limited only by the **RESERV** value.

4.15 Redefinition

In order to protect you from errors resulting from the unintentional reuse of a symbol, FIL normally diagnoses an error when a **geometric** or **text** symbol is reused.

Example:

```
P1 = POINT / 3, 3, 10
-
-
-
P1 = POINT / 7, 5, 10
```

The second statement produces a diagnostic because the symbol **P1** is being reused. In some cases, however, you may deliberately want to reuse symbols. You may use the **REDEF** statement to indicate whether redefinition is permitted or not.

4.15.1 REDEF / ON-OFF

ON indicates that symbol redefinition is permitted. **OFF** indicates symbol redefinition is not permitted and is the assumed setting if no **REDEF** statement is given.

***Example*:**

```
REDEF/ON              $$ ALLOW REDEFINITION
P1 = POINT / 3, 4, 10
GOTO / P1
-
-
P1 = POINT / 4, 5, 10
GOTO / P1
```

A symbol references the last preceding values assigned to it. In the preceding example, the first **GOTO / P1** moves the cutter to (**3, 4, 10**); the second, to (**4, 5, 10**).

REDEF does not apply to scalar symbols, which may always be redefined regardless of the status of **REDEF**.

A geometry symbol and a scalar can now be redefined (switched) with **REDEF/ON**. Previously this would cause an Error to be produced.

Example:

```
REDEF/ON
D1 = DATA/1,2.3       $$ DATA STATEMENT
D1 = 3.500            $$ SCALAR
D1 = TEXT/'1234'      $$ TEXT
```

4.16 Geometric Definitions

Geometric definition statements are used to define CL points and vectors within FIL for execution by the post processor. The general format of a geometric definition statement is:

name = typeldefinition list

In the above format, ***name*** is the symbol assigned to the entry being defined; ***type*** indicates the geometric type - **POINT** or **VECTOR ;** and the ***definition list*** is a series of scalars, vocabulary words, or symbols that comprise the details of the definition.

4.16.1 The POINT Definition

4.16.1.1 A POINT is a unique position in space.

name **= POINT /** ***x, y, z***

In the above definition, ***name*** is the symbol being assigned to the point definition, **POINT** is the definition type, and ***x, y, z*** are scalars or scalar symbols defining the coordinates of the point.

Example:

```
P1 = POINT / 4, 6, 20
P2 = POINT / 4, 4, 0
P3 = POINT/ 6, 7, 10
```

4.16.1.2 A POINT multiplied by a pre-defined MATRIX.

name **= POINT /** ***point*****, MODIFY,*****matrix***

In the above definition, ***name*** is the symbol being assigned to the point definition, **POINT** is the definition type, ***point*** is a pre-defined point symbol, **MODIFY** is the command modifier and ***matrix*** is a predefined MATRIX symbol.

Example:

```
P1 = POINT / 1,2,3
M1 = MATRIX/TRANSL,10,10,10
P2 = POINT/P1,MODIFY,M1              $$ multiply P1 by matrix M1
```

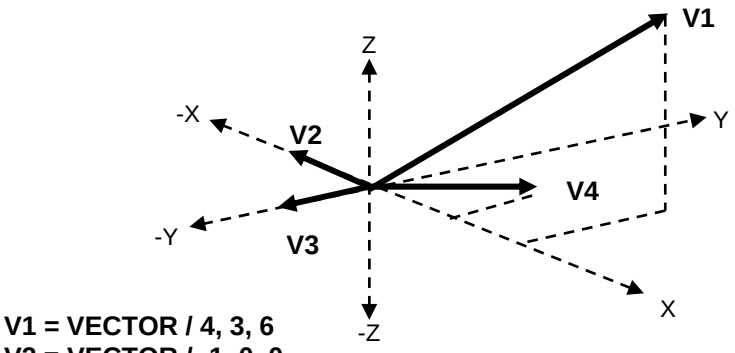
4.16.2 The VECTOR Definitions

A **VECTOR** is a geometric entity that has both magnitude and direction. The VECTOR is used in FIL and by the post processor to define a tool axis. The following methods of defining a vector may be used in FIL.

4.16.2.1 Vector Defined by Components

- **VECTOR/ x, y, z**

This format defines a vector in terms of its components along the x, **y,** and z-axes.

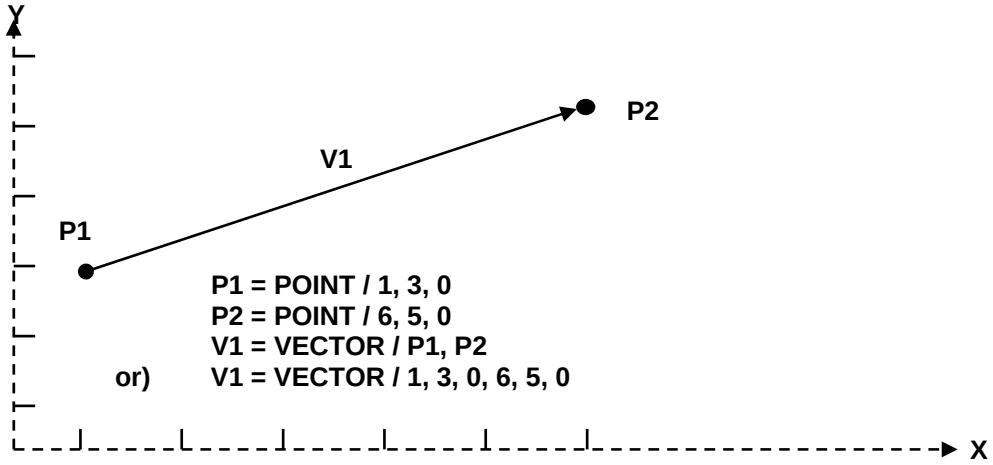


V1 = VECTOR / 4, 3, 6
V2 = VECTOR / -1, 0, 0
V3 = VECTOR / 0, -1, 0
V4 = VECTOR / 2, 2, 0

4.16.2.2 Vector Defined Through Two Points

- **VECTOR/ x1, y1, z1, x2, y2, z2**
- **VECTOR/ point, point**

These formats define a vector from the first specified point to the second.

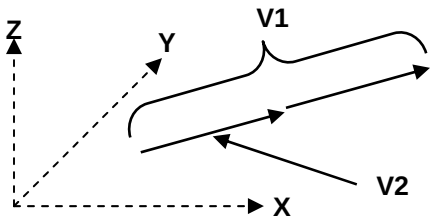


P1 = POINT / 1, 3, 0
P2 = POINT / 6, 5, 0
V1 = VECTOR / P1, P2
or) V1 = VECTOR / 1, 3, 0, 6, 5, 0

4.16.2.3 Vector Defined as a Scalar Times a Vector

- **VECTOR / scalar, TIMES, point-vector**

This format defines a vector as a scalar times a vector or a point. A point is treated like a vector from the origin to the point.

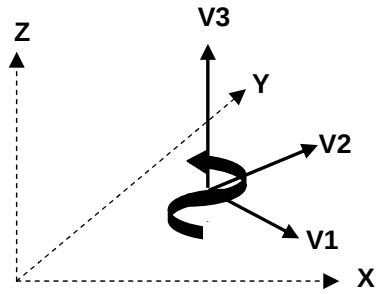


V1 = VECTOR / 2.6, TIMES, V2

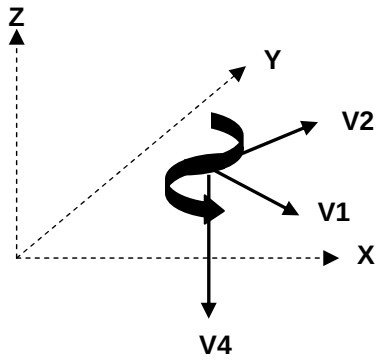
4.16.2.4 Vector Defined as the Cross Product of Two Vectors

- **VECTOR / vector-point, CROSS, vector-point**

This format defines a vector as the cross product of two vectors. The resultant vector is perpendicular to the plane of the two given vectors and has a length equal to the product of the lengths of the given vectors times the sine of the angle between them. The direction of the vector is the direction of advance caused by rotating a right hand thread from the first given vector to the second. A point is treated like a vector from the origin to the point.



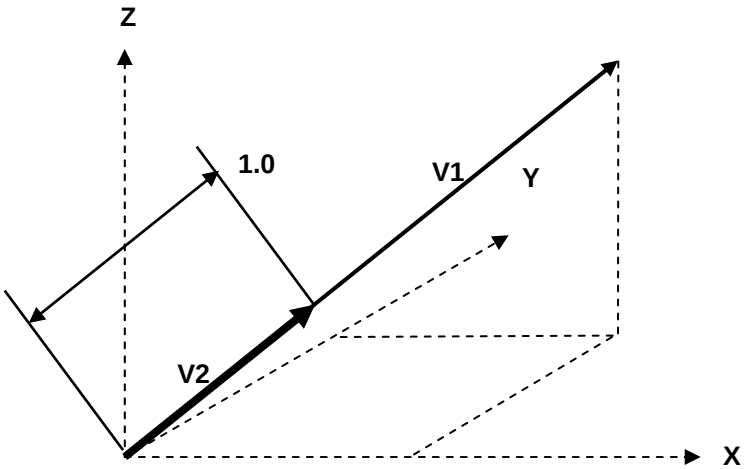
V3 = VECTOR / V1, CROSS, V2
V4 = VECTOR / V2, CROSS, V1



4.16.2.5 Vector Defined by Normalizing a Vector

- **VECTOR / UNIT, vector-point**
- **VECTOR / UNIT, x, y, z**

This format defines a vector by normalizing a given vector. The normalized vector has the same direction as the original but has a magnitude of 1. A point is treated like a vector from the origin to the point. An error results if all three components of the given vector or point are zero.

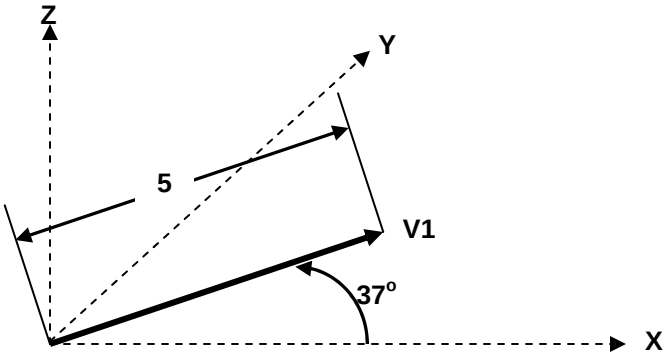


V1 = VECTOR / 4, 4, 4
V2 = VECTOR / UNIT, V1
or) V2 = VECTOR / UNIT, 4, 4, 4

4.16.2.6 Vector Defined by Its Length and an Angle

- **VECTOR / LENGTH, length, ATANGL, angle, XYPLAN-YZPLAN-ZXPLAN**

This format defines a vector by its length and an angle in a coordinate plane. The angle is measured from the first axis specified in the modifier (x-axis for XYPLAN, y-axis for YZPLAN, z-axis for ZXPLAN) and is positive for counterclockwise measurement from the axis to the vector, negative for clockwise.

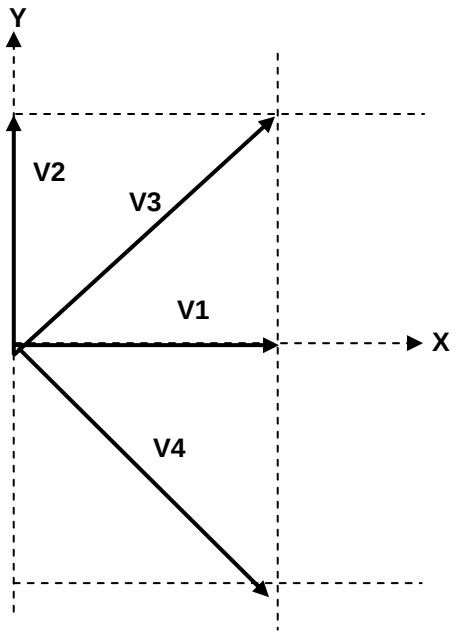


V1 = VECTOR / LENGTH, 5, ATANGL, 37, XYPLAN

4.16.2.7 Vector Defined as the Sum or Difference of Two Vectors

- **VECTOR / vector-point, PLUS-MINUS, vector-point**

This format defines a vector as the sum or difference of two vectors or points. A point is treated like a vector from the origin to the point.




```
V1 = VECTOR / 1, 0, 0
V2 = VECTOR / 0, 1, 0
V3 = VECTOR / V1, PLUS, V2          $$ Result (1,1,0)
V4 = VECTOR / V1, MINUS, V2          _____ $$ Result (1,-1,0)
```

4.16.2.8 A VECTOR multiplied by a pre-defined MATRIX.

name **= VECTOR /** ***vector*****, MODIFY,*****matrix***

In the above definition, ***name*** is the symbol being assigned to the vector definition, **VECTOR** is the definition type, ***vector*** is a pre-defined vector symbol, **MODIFY** is the command modifier and ***matrix*** is a predefined MATRIX symbol.

Example:

```
V1 = VECTOR / 1,2,3
M1 = MATRIX/TRANSL,10,10,10
V2 = VECTOR/V1,MODIFY,M1                $$ multiply V1 by matrix M1
```

4.16.3 The MATRIX Definitions

A matrix is a rectangular array of numbers. The FIL word MATRIX refers to a particular kind of matrix having three rows and four columns that is used to transform geometry from one coordinate system to another.

An APT matrix can be written using matrix notation as follows:

```
a1    b1    c1    d1
a2    b2    c2    d2
a3    b3    c3    d3
```

A matrix can be used to transform geometry from an auxiliary coordinate system to a base coordinate system. In the matrix specified above, the elements in the first column (a1, a2, a3) are the components of a vector, relative to the base system which represents the positive x-axis of the auxiliary system. Similarly, the second column (bl, b2, b3) defines the y-axis and the third column (c1, c2, c3) defines the z-axis. The fourth column (dl, d2, d3) defines the origin of the auxiliary system relative to the base system.

The auxiliary axis vectors, columns 1, 2, and 3, define the rotation generated by the matrix, while the auxiliary origin, column 4, defines the translation.

The significance of the twelve elements of a matrix may be illustrated by a demonstration of the way they are used to transform a point. Suppose a given point has coordinates (x, y, z) relative to an auxiliary coordinate system. The point can be transformed to a point with coordinates (xt, yt, zt) relative to the base system as follows:

```
xt = a1x + b1y + c1z + d1
yt = a2x + b2y + c2z + d2
zt = a3x + b3y + c3z + d3
```

The matrix vectors (columns 1, 2, and 3) must be mutually perpendicular unit vectors in order for correct results to be produced. All available methods of defining a matrix produce such vectors except the format that defines a matrix directly in terms of its twelve elements and the scale matrix format. You can also use the matrix data for the G-Post 12-parameter TRANS/cmd as below:

Example:

```
M1 = MATRIX/TRANSL,10,5,0
TRANS / (DATA/M1)              $$ nested () DATA must be specified
```

4.16.3.1 Matrix Defined by Its 12 Elements

- **MATRIX / a1, b1, c1, d1, a2, b2, c2, d2, a3, b3, c3, d3**

This format defines a matrix directly in terms of its 12 elements.

M1 = MATRIX / 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0

This is the so called identity matrix which defines the auxiliary and base systems to be identical. Application of this matrix causes no transformation to take place.

M2 = MATRIX / 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0

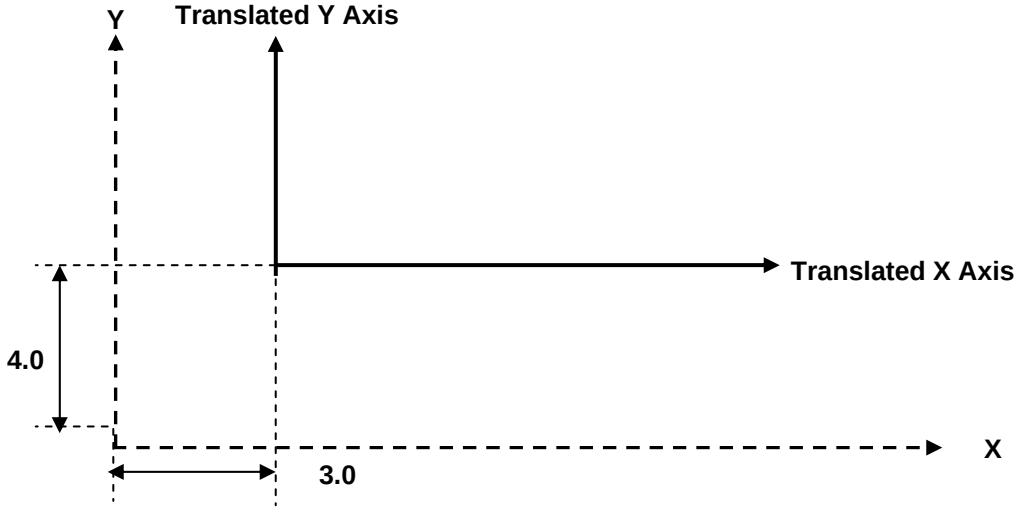
In this matrix, the first and third rows of the identity matrix are interchanged, causing the x and z axes of the two systems to be interchanged.

4.16.3.2 Matrix Defined as a Translation

- **MATRIX / TRANSL, d1, d2, d3**

This format defines a matrix as a translation only. The z translation, d3, may be omitted in which case it is assumed to be zero. The values d1, d2, d3 are coordinates of the origin of the auxiliary system relative to the base system. This format generates the following matrix:

1	0	0	d1
0	1	0	d2
0	0	1	d3



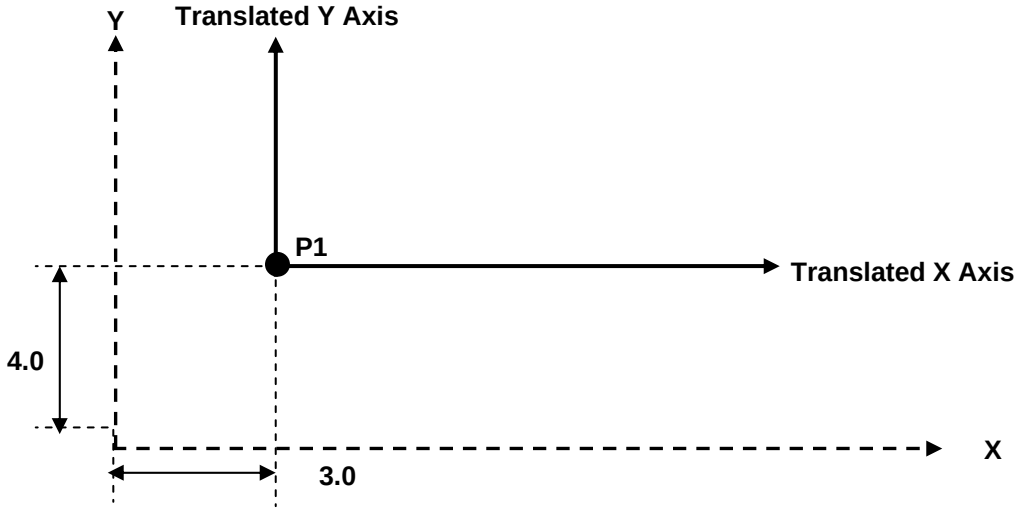
M1 = MATRIX/ TRANSL, 3, 4, 0

The point (0, 0, 0) in the auxiliary system is (3, 4, 0) in the base system.

- **MATRIX/ TRANSL, point-vector**

This format defines a matrix as a translation only. The x,y,z or a,b,c coordinates or the point or vector are the coordinates of the origin of the auxiliary system relative to the base system. This format generates the following matrix:

1	0	0	point x or vector a
0	1	0	point y or vector b
0	0	1	point z or vector c



P1 = POINT / 3, 4, 0
M1 = MATRIX/ TRANSL, P1

The point (0, 0, 0) in the auxiliary system is (3, 4, 0) in the base system.

4.16.3.3 Matrix Defined as a Rotation

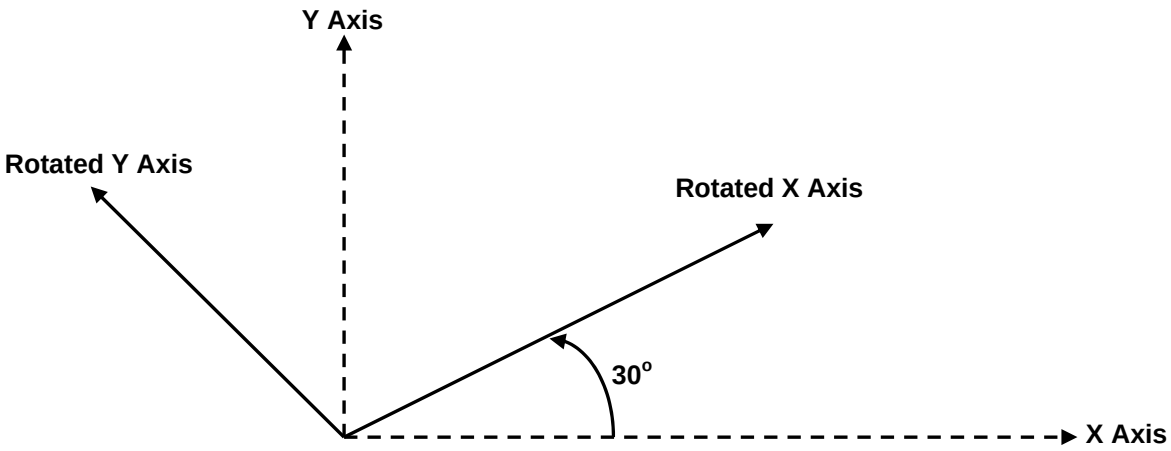
XYROT
- **MATRIX / YZROT, angle**
ZXROT

This format defines a matrix as a rotation only in the specified coordinate plane. The angle is the one through which the axes of the base coordinate system must be rotated to make them parallel to those of the auxiliary system.

XYROT designates rotation about the z-axis. When viewed from the positive z-axis looking toward the origin, the angle of rotation is measured from the positive x-axis and is positive for counterclockwise measurement, negative for clockwise.

The following matrix is generated:

cos angle	-sin angle	0	0
sin angle	cos angle	0	0
0	0	1	0

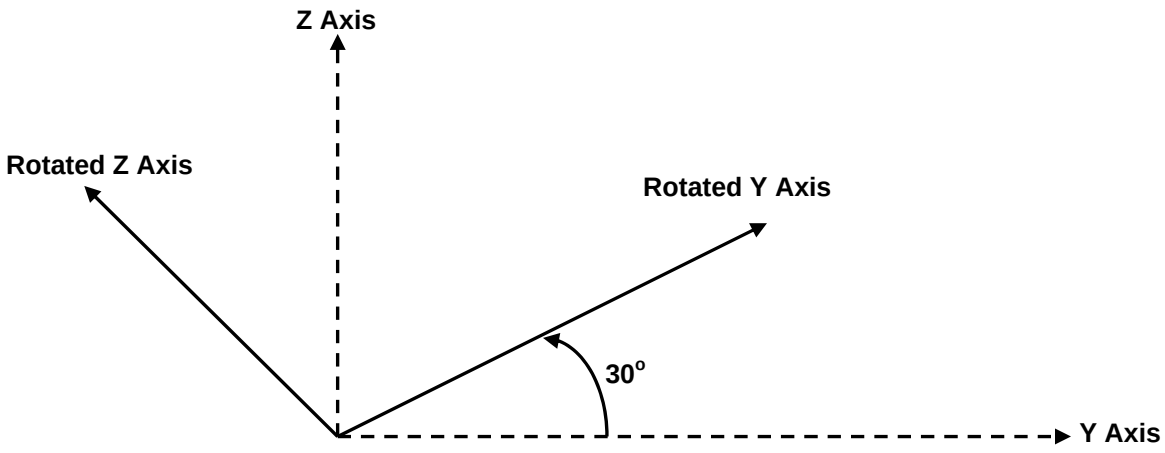


M2 = MATRIX / XYROT, 30

YZROT designates rotation about the x-axis. When viewed from the positive x-axis looking toward the origin, the angle of rotation is measured from the positive y-axis and is positive for counterclockwise measurement, negative for clockwise.

The following matrix is generated:

1	0	0	0
0	cos angle	-sin angle	0
0	sin angle	cos angle	0

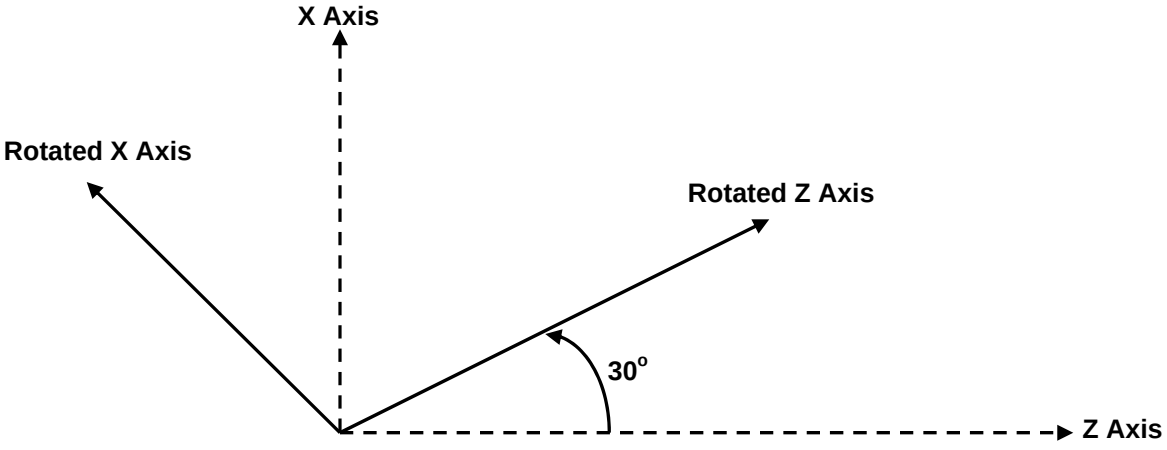


M3 = MATRIX / YZROT, 30

ZXROT designates rotation about the y-axis. When viewed from the positive y-axis looking toward the origin, the angle of rotation is measured from the positive z-axis and is positive for counterclockwise measurement, negative for clockwise.

The following matrix is generated:

cos angle	0	sin angle	0
0	1	0	0
-sin angle	0	cos angle	0



M4 = MATRIX / ZXROT, 30

4.16.3.4 Matrix Defined by a Scale Factor

- **MATRIX / SCALE, s**

This format defines a matrix as a scale factor. The following matrix is generated:

s	0	0	0
0	**s**	0	0
0	0	**s**	0

Unless ***s*** equals 1, this matrix does not have unit rotation vectors and should therefore be used with **CAUTION.** Its primary purpose is to scale GOTO points with TRANS.

Example:

```
M3 = MATRIX / SCALE, .5
TRANS / (DATA/M3)
GOTO/ 30, 10, 4
```

The TRANS operation, using the scale matrix, converts the point (30, 10, 4) to (15, 5, 2), which is half scale.

4.16.3.5 Matrix Defined as a Matrix Product

- **MATRIX / matrix
XYROT, angle
YZROT, angle
ZXROT, angle
TRANSL, d1, d2, d3,
SCALE, s**
  **matrix
XYROT, angle
YZROT, angle
ZXROT, angle
TRANSL, d1, d2, d3
SCALE, s**

This format defines a matrix as the product of two matrices. Applying a product matrix to geometry has the effect of transforming it twice, first with one of the component matrices, then with the other. The order in which the matrices are specified is important since, in general, if A and B are matrices, then A times B does not equal B times A.

If each of the two matrices is specified by a matrix symbol or by a nested matrix definition, the order of multiplication is from right to left.

```
M4 = MATRIX / M1, M2
```

This means M2 times M1; that is, M4 is defined to be a matrix that has the combined effect of applying the transformation defined by M2 first, then that defined by M1.

When either of the two matrices or both are specified by formats other than a matrix symbol or a nested matrix definition, the order of multiplication is from left to right.

```
M1 = MATRIX / XYROT, 30, TRANSL, 3, 4, 0
```

The 30-degree rotation is applied first, then the translation.

4.16.3.6 Matrix Defined as the Inverse of a Matrix

- **MATRIX / INVERS, matrix**

This format generates a matrix that performs the inverse transformation of that performed by the given matrix.

```
M5 = MATRIX / TRANSL, 2, 5, 0
M6 = MATRIX  /INVERS, M5
```

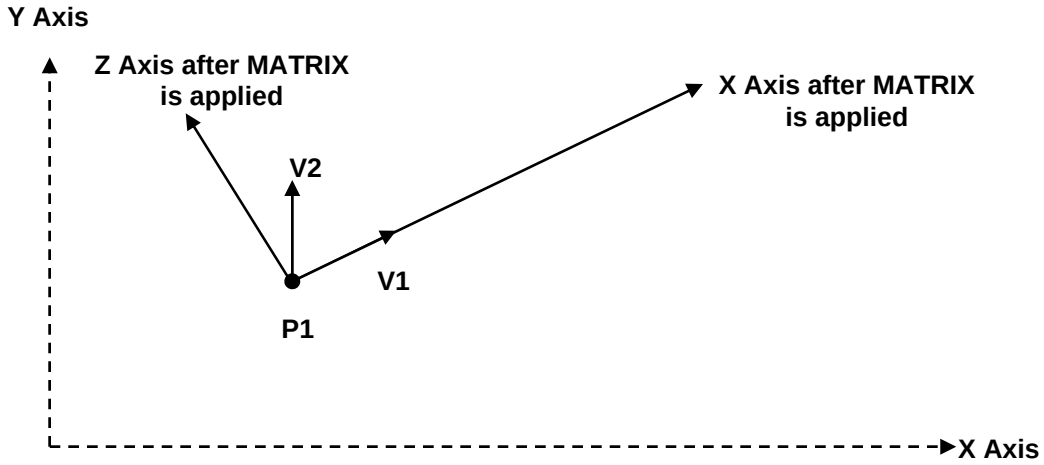
If M5 transforms data from system A to system B, then M6 transforms geometry from system B to system A. In this particular case, M6 is equivalent to:

M6=MATRIX/TRANSL, -2,-5, 0

4.16.3.7 Matrix Defined by a Point and Two Vectors

- **MATRIX / point, vector, vector**

This format defines a matrix in terms of a point and two vectors. The point is the origin of the a system relative to the base system. The first vector represents the positive x-axis of the auxiliary system. The second vector should point from the given point toward the first or second quadrant of the auxiliary system. It is used to select the desired orientation of the two possible orientations of the auxiliary y the one that makes an acute angle with the second vector.




```
P1 = POINT / 3, 4
V1 = VECTOR / LENGTH, 1, ATANGL, 30, XYPLAN
V2 = VECTOR / 0, 1, 0
M7 = MATRIX / P1, V1, V2
```

4.16.3.8 Matrix Defined by a Point, Vector and an Angle

- **MATRIX/point, vector, angle**

This definition of a MATRIX is to be used for rotating a geometric surface about a point, using a vector from the point as the axis of rotation.

The rotation angle is specified in degrees. The direction of rotation is determined by looking from the point along the axis of the vector. A positive angle will result in a clockwise rotation.

The MATRIX definitions in the following example would yield the same result since in one instance the direction of viewing is from the negative X axis and the other is from the positive X axis.

```
V1 = VECTOR / 1,0,0
P1 = POINT / 0,0,0
M1 = MATRIX / P1,V1,30
M2 = MATRIX / YZROT,30
TRANS / (DATA/M1)    $$ (or M2)
```

4.16.3.9 Matrix Defined as a Mirror Image

- **MATRIX/MIRROR, modifier 1, modifier 2, modifier 3**

This format defines a mirror image about one or more coordinate axes. One, two or three modifiers may be specified. The allowable modifiers are XYPLAN, YZPLAN, and ZXPLAN.

XYPLAN designates a mirror image relative to the z-axis; YZPLAN relative to the x-axis; and ZXPLAN relative to the y-axis. When more than one modifier is specified the order is irrelevant.

Example:

The following illustrates the results when various mirror matrices are applied to the point (5, 9, 11):

```
M8A = MATRIX / MIRROR, XYPLAN                    (5, 9, -11)
M8B = MATRIX / MIRROR, XYPLAN, ZXPLAN            (5, -9, -11)
M8C = MATRIX / MIRROR, XYPLAN, YZPLAN, ZXPLAN    (-5, -9, -11)
M8D = MATRIX / MIRROR, ZXPLAN                    (5, -9, 11)
```

4.17 Motion Statements

4.17.1 The FROM Statement

This statement specifies the initial location of the cutter. It can have the following format:

FROM / *point*	**$$ Non-MULTAX CL**
FROM / *x, y, z*	**$$ Non-MULTAX CL**
FROM / *point, vector*	**$$ MULTAX CL only**
FROM / *x, y, z, l, j, k*	**$$ MULTAX CL only**

The point may be specified by the symbol for a ***point*** or by its ***x,y, z*** coordinates.

When programming **FROM** it is important to know if the post processor is in multi-axis mode.

When in multi-axis mode the **FROM** must contain a ***point*** and a ***vector*** specification. The point may be specified by the symbol for a ***point*** or by its ***x,y, z*** coordinates. The vector may be specified by a symbol for a ***vector*** or by its ***x, y, z*** coordinates.

Example:

Non-Multi-axis **FROM:**

FROM / 1.5, 20.0, 10.5
FROM / P1

Multi-axis **FROM:**

FROM / P1, V1
FROM / 1.7, 30.5, 10.0, 0, 1, 0

4.17.2 The GOTO Statement

This statement moves the CL point to the designated point. The **GOTO** formats are identical to the **FROM** formats:

GOTO / *point*	**$$ Non-MULTAX CL**
GOTO / *x, y, z*	**$$ Non-MULTAX CL**
GOTO / *point, vector*	**$$ MULTAX CL only**
GOTO / *x, y, z, l, j, k*	**$$ MULTAX CL only**

The point may be specified by the symbol for a ***point*** or by its ***x,y, z*** coordinates.

When programming **GOTO** it is important to know if the post processor is in multi-axis mode.

When in multi-axis mode the **GOTO** must contain a ***point*** and a ***vector*** specification. The point may be specified by the symbol for a ***point*** or by its ***x,y, z*** coordinates. The vector may be specified by a symbol for a ***vector*** or by its ***x, y, z*** coordinates.

Example:

Non Multi-axis **GOTO:**

GOTO / 2.1, 2.0, 15.5
GOTO / P6

Multi-axis **GOTO:**

GOTO / P2, V3
GOTO / 1.0, 25.4, 11.5, 1, 0, 0

4.17.3 The GODLTA Statement

The **GODLTA** (go delta) statement specifies incremental values that are to be added to the coordinates of the current cutter location to obtain the new cutter location and to move a delta distance along the tool axis from the current point. In this way, it specifies the new cutter location in terms of incremental distances from the previous location rather than by absolute coordinates. **GODLTA** has the following formats:

GODLTA / *dx, dy, dz*
GODLTA / *vector*
GODLTA / d

With the format ***dx, dy, dz,*** the incremental distances are explicitly specified. With the ***vector*** format, the components of the designated vector are used.

Example:

GODLTA / V3
GODLTA / 1.0, 4.0, 11.5
GODLTA / .5, .5, 0
GODLTA / 1, 1, 1

4.18 CANON Definitions

In addition to the regular methods that are available for defining geometric entities, a **CANON** definition may be used to define an entity directly in terms of its canonical elements or in terms of modifications to be made to a previously created canonical form.

A **CANON** definition always permits redefinition of a symbol even though redefinition has not been enabled (by REDEF/ON) for regular definitions.

Type* / *symbol*, CANON, *canonical elements

In this format, ***type*** is a major word indicating the canonical form type - **POINT**, or **VECTOR**.

The ***symbol*** represents a previously defined entity of the same type.

The canonical form denoted by the symbol serves as the base canonical form for the new definition. Values are given for ***canonical elements*** that are to be different from those in the base canonical form.

Values replace elements on a positioned basis - the value in the first position replaces the first element in the base canonical form, etc. A comma is used to space over a position where the element is to retain its base value. Trailing commas following the last specified value are unnecessary but are permissible.

Examples:

```
P1=POINT/6,4,8
P2=POINT/P1, CANON,7        (P2 = 7, 4, 8)
P3=POINT/P1, CANON,,9       (P3 = 6, 9, 8)
P4=POINT/P1, CANON,,,3      (P4 = 6, 4, 3)
```

4.19 The DATA Statement

With **DATA**, you can input all the data items that vary from one use of the program to another in a single statement.

The format of the **DATA** statement is:

symbol* = DATA / *entry 1, entry 2,--, entry n

The minimum number of ***entries*** allowed is one; the maximum is 82.

Each ***entry*** can be a scalar, a vocabulary word, or a previously defined symbol.

When an ***entry*** is the symbol for a canonical form, the elements of the referenced canonical form replace the symbol in the **DATA** canonical form. Thus, a **DATA** statement generates a canonical form composed of scalars and/or vocabulary words.

Examples:

```
D1 = DATA / 9, 10, 11          $$ canonical form is 9, 10, 11
D2 = DATA / 15, IPM            $$ canonical form is 15, IPM
D3 = DATA / D1, 20, 21         $$ canonical form is 9, 10, 11,
                               $$ 20, 21
```

The appropriate names can be assigned to the scalars in a **DATA** canonical form by using **OBTAIN** or **CANF**.

For example, suppose that a macro is written to handle similar configurations, the only difference among the members being that six particular dimensions can vary. These dimensions are referenced in the body of the macro by the symbols **A, B, C, D, E, F.**

The dimensions for a particular configuration are input in a **DATA** statement:

```
D 1 = DATA / 1.5, 3.75, 2.5, 5, 6.2,10
```

An **OBTAIN** statement then sets the symbols to the values of the **DATA** canonical elements:

```
OBTAIN, DATA / D1, A, B, C, D, E, F
```

In some applications, it is desirable to be able to vary the number of elements in a **DATA** statement from one use to another. FIL permits this by not regarding it as an error if more symbols are included in the

OBTAIN statement than there are elements in the **DATA** canonical form. The excess symbols are simply ignored.

An alternative method of using **DATA** is provided as follows: A **DATA** symbol can be referenced to the right of the slash in a statement where a series of scalars and/or vocabulary words would normally appear. The effect is the same as if the elements of the referenced **DATA** canonical form appeared explicitly in place of the symbol.

Examples:

```
D1= DATA/1, 2, 3
P1=POINT/D1

is equivalent to
P1= POINT/1, 2, 3

**********
SD= DATA/200, RPM, CLW
SPINDL/SD

is equivalent to
SPINDL/200, RPM, CLW

**********
FD=DATA/.1, IPR
FEDRAT/FD

is equivalent to
FEDRAT/.1, IPR

**********

RESERV / D, 4
D(1)=DATA/500,SFM
D(2)=DATA/CLW
D(3)=DATA/RANGE,2
D(4)=DATA/MAXRPM,3000

SPINDL/D(1, THRU, 4)

is equivalent to
SPINDL/500,SFM,CLW,RANGE,2,MAXRPM,3000
```

You can indicate that only selected elements of the **DATA** canonical form are to be used by specifying the following after the **DATA** symbol:

RANGE, *m*, *n*

m is the number of the first element to be used and ***n*** is the number of the last.

Example:

```
D1 = DATA/1, 5, 6.5, 10, IPM, 300
FEDRAT/D1, RANGE,4,5

is equivalent to
FEDRAT/10, IPM

**********
SPINDL/D1, RANGE, 6,6

is equivalent to
SPINDL/300
```

4.20 The OBTAIN Statement

The **OBTAIN** statement may be used to extract scalars from a canonical form and assign symbols to them. Its format is:

OBTAIN, ***type/canon symbol, scalar symbols***

Type is a major word indicating the canonical form type - **POINT**, **VECTOR**, or **DATA.**

Canon symbol is the symbol of a canonical form of the specified type.

The ***scalar symbols*** are the symbols to be assigned to the elements of the canonical form. The number of scalar symbols can be as small as one and as large as the number of elements in the canonical form.

Symbols and elements are matched according to position - the symbol in the first position is assigned to the first element, etc. It is not necessary, however, to assign a symbol to every element. A comma can be used to space over a position whose corresponding element is not to be assigned a symbol. Trailing commas following the last symbol are not necessary but are permissible.

Use of this feature requires a knowledge of the order and the significance of the elements of canonical forms.

Examples:

```
P1 = POINT/3, 8, 5
OBTAIN, POINT/P1, A, B, C     $$ (A=3, B=8, C=5)
OBTAIN, POINT/P1, A           $$ (A=3)
OBTAIN, POINT/P1, A,,$$ (A=3)
OBTAIN, POINT/P1,,B           $$ (B=8)
OBTAIN, POINT/P1,,B,C         $$ (B=8, C=5)
OBTAIN, POINT/Pl,,,C          $$ (C=5)
```

4.21 Functions

This section will explain the different functions available within FIL. These functions return a value or TEXT string to the specified variable. There are additional computation functions explained previously in this chapter as well as one additional function, **POSTF**. The **POSTF** function is explained in Chapter 5.

Note:	In the syntax description, we use the variable ***rslt***. You can use any variable you want, but be cautious about this. In the instance where zero is returned, be sure to use a variable that you will not be using elsewhere. Remember to use **redef/on** if you deliberately want to redefine text variables.

4.21.1 The CANF Function

***rslt* = CANF(*t1*,1)**

rslt: Number of characters in ***t1***

t1: Text string

1: This value is always 1 and instructs the **CANF** to return the number of characters. This functions differs from the **CANF** described earlier in the computation section of this chapter.

CANF finds the number of characters in a string.

Example:

```
T1 = TEXT / 'THIS IS A TEXT STRING TEST'
NCH = CANF(T1,1)      $$ NCH = 26
```

4.21.2 The CMPRF Function

***rslt* = CMPRF(*t1*,*t2*)**

rslt: Result of match attempt

= 0 No match
= 1 Match

t1: Text string to be compared

t2: Text string to be compared

CMPRF compares two text strings.

Example:

```
TA = TEXT/ 'ABCDE'
TB = TEXT/ 'XYZ123'
TC = TEXT/ 'ABCDE'
MAT1 = CMPRF(TA,TB)      $$ Strings do not match
                         $$ MAT1 = 0
MAT2 = CMPRF(TA,TC)      $$ Strings match; MAT2 = 1
```

4.21.3 The FILEF Function

***rslt* = FILEF(*fn,op,t1*)**

rslt : Result of operation

- **= 0** Requested file operation succeeded - OK
- **= 1** Requested file operation failed - error

fn: File number to be used (0, 1, 2, 3, or 4)

- **= 0** Terminal, read/write only,
- **= 1** Any ASCII text file.
- **= 2** Any ASCII text file.
- **= 3** .LST file, you can only write to this file. Do not OPEN or CLOSE this file.
- **= 4** .PUn file, you can only write to this file. Do not OPEN or CLOSE this file.

op: Operation to be executed

- **= 1** Write one record from text string ***t1*** to file ***fn***.
- **= 2** Open an existing file; file name is ***t1; fn*** must be 1 or 2.
- **= 3** Open a new file; file name in ***t1; fn*** must be 1 or 2.
- **= 4** Rewind file; ***t1*** not required; ***fn*** must be 1 or 2.
- **= 5** Close file; ***t1*** not required; ***fn*** must be 1 or 2.
- **= 6** Close and delete file; ***t1*** not required; ***fn*** must be 1 or 2.
- **= 7** Inquire if file exist; ***rslt*** will be set to 1 if file exist, otherwise ***rslt*** will be set to 0; ***fn*** must be 1 or 2.

t1: Text string

FILEF operates on external files. You can have a maximum of two files active at any given time. If ***fn*** is 0, output is sent to the terminal.

For terminal I/O, the first character in ***t1*** is assumed to be the FORTRAN carriage return format for the line. These formats are shown below:

b	Carriage return/line feed (single-spaced line)
0	Carriage return/line feed, carriage return/line feed (double-spaced line)
1	Form feed
$	No carriage return/no line feed

To read a record from a file, use the command **TEXT/READ,*fn*** as described later in this chapter.

Note: When using **FILEF(4,1,*t1*)** to write directly to the MCD (PUn) file, there is an additional (4th) parameter that can be used to automatically append the current sequence number to the ***t1*** string. Add a 1 as the 4th parameter to activate the output of the sequence number with the ***t1*** string. The syntax for this command is:

***rslt* = FILEF(4,1,*t1*,1)**

FILEF Examples:

To open an existing file;

```
FN = TEXT / 'TEST.PPR'          $$ THE FILE NAME
FEX = FILEF(1,7,FN)             $$ CHECK TO SEE IF FILE
                                $$ EXIST

IF(FEX .EQ. 1)THEN
  DMY = FILEF(1,2,FN)           $$ OPEN EXISTING FILE FN AS
                                $$ LOGICAL UNIT 1

ELSE
  FOPEN = FILEF(1,3,FN)         $$ FILE DID NOT EXIST, OPEN A
                                $$ NEW FILE NAMED FN AS
                                $$ LOGICAL UNIT 1
ENDIF
```

To open an new file;

```
FN = TEXT / 'TEST.PPR'          $$ THE FILE NAME
FEX = FILEF(1,7,FN)             $$ CHECK TO SEE IF FILE
                                $$ EXIST

IF(FEX .EQ. 1)THEN
  DMY = FILEF(1,2,FN)           $$ OPEN EXISTING FILE FN AS
                                $$ LOGICAL UNIT 1
  DMY = FILEF(1,6)              $$ CLOSE AND DELETE
                                $$ LOGICAL UNIT 1 (FILE FN)
ENDIF
FOPEN = FILEF(1,3,FN)           $$ OPEN A NEW FILE NAMED
                                $$ FN AS LOGICAL UNIT 1
```

To close and save an opened file;

```
IF(FOPEN .EQ. 0)THEN            $$ FOPEN IS THE VARIABLE
                                $$ SET DURING THE FILE OPEN
                                $$ COMMAND.
  DMY = FILEF(1,5)              $$ CLOSE LOGICAL UNIT 1 AND SAVE
ENDIF
```

To close and delete an opened file;

```
IF(FOPEN .EQ. 0)THEN            $$ FOPEN IS THE VARIABLE
                                $$ SET DURING THE FILE OPEN
                                $$ COMMAND.
  DMY = FILEF(1,6)              $$ CLOSE LOGICAL UNIT 1 AND  DELETE
ENDIF
```

To write a text string to an opened file;

```
IF(FOPEN .EQ. 0)THEN                    $$ FOPEN IS THE VARIABLE
                                        $$ SET DURING THE FILE OPEN
                                        $$ COMMAND.
  T1 = TEXT / ‘ THIS IS TEXT’           $$ A TEXT STRING
  DMY = FILEF(1,1,T1)                   $$ WRITE STRING T1 TO
                                        $$ LOGICAL UNIT 1
ENDIF
```

To write a text string to the terminal;

```
T1 = TEXT / ‘ENTER TOOL NUMBER -> ‘ $$ A TEXT STRING
MY = FILEF(0,1,T1)                      $$ WRITE STRING T1 TO
                                        $$ LOGICAL UNIT 0, THE
                                        $$ TERMINAL
```

4.21.4 The ICHARF Function

***rslt*=ICHARF(*t1*)**

rslt: Number representing a character in the **ASCII** table.

T1: Text string.

ICHARF Converts a text string to a numeric value. Only the first character of ***T1*** is used.

Example:

```
T1 = TEXT/ ‘THIS IS TEST #’
T2 =TEXT / 'A'
RSLT = ICHARF(T2)                       $$ RSLT = 65 (ASCII A)
T3 = TEXT / ‘THIS IS TEST # ’,RSLT
$$ T3 = THIS IS TEST # 65
```

4.21.5 The ICODEF Function

***rslt* = ICODEF(*word*)**

rslt: Scalar: integer code assigned to the ***word***
word: Any post processor vocabulary word or symbol

ICODEF() will allow a symbol as the argument. If the symbol is not defined, the function will return -1. This can be used to determine if a symbol has been defined.

You can perform the opposite function (return the vocabulary word for an integer code) by using the FIL command **CONVI**. Let's use our favorite example, **SPINDL/300,CLW,** to illustrate what we mean. Assume that we knew that the integer code of the minor word **CLW** was 60, but we'd forgotten exactly what the word was. Usually, we'd pull out our G-Post User's Guide and look up the word. However, we can't do that today because Fred (our fishing buddy) borrowed our book and didn't return it. No problem. What we'll do is enter the following code:

```
SPINDL/300,CONVI,60
```

FIL converts the integer code, 60, to the correct minor word, **CLW**, and we can take a break from all this strenuous work.

Example:

```
IWD = POSTF(7,4)                        $$ See POSTF chapter
IF( IWD .EQ. (ICODEF(ON))) THEN         $$ Check word for ON
  PPRINT/'TURN ON COOLANT'
ENDIF

************************

SPCOD = POSTF(7,4)                      $$ SEE POSTF CHAPTER

CASE / SPCOD
        WHEN/ (ICODEF(ON))              $$ SPINDL/ON
          PPRINT / 'SPINDLE ON'

        WHEN/ (ICODEF(OFF))$$ SPINDL/OFF
          PPRINT/ ' SPINDLE OFF'

        WHEN/ (ICODEF(RPM))             $$ SPINDL/RPM,nnn
          PPRINT/ ' SPINDLE SET TO RPM '

        WHEN/ OTHERS                    $$ ERROR OUT
          PPRINT/ ' INVALID SPINDLE COMMAND '
ENDCAS

********************

4TH = POSTF(7,4)                        $$ See POSTF chapter
IF( 4TH .EQ. (ICODEF(FLOOD))) THEN
  PPRINT/'TURN FLOOD COOLANT ON'
ENDIF
```

Example using a symbol:

```
IWD =   ICODEF(SPINDL)          $$ Will return 1031, the integer code.
ICK =   ICODEF(HPPT)            $$ Will return -1 since HPPT is not defined
FED =   50                      $$ A scalar definition.
ICK =   ICODEF(FED)             $$ Will return 0, for SCALAR type
HPT =   POINT/0,0,0             $$ A point definition
ICK =   ICODEF(HPT)             $$ Will return 1, for POINT type
VC1 =   VECTOR/0,1,0            $$ A vector definition
ICK =   ICODEF(VC1)             $$ Will return 11, for VECTOR type
```

4.21.6 The INDXF Function

rslt* = INDXF(*t1*,*t2[,arg3])

rslt: Result of text string match attempt
= 0 No text string match
= n Text string matched, location of the first character
t1: Major text string
t2: Text string to match
arg3: Optional, by setting this argument to 1 INDXF will locate the last matching string

INDXF locates a the first occurrence of a text string within a major text string, arg3=1 changes this to the last occurrence.

Example:

```
TA = TEXT/ 'ABCDE'                    $$ Locate C in the string ABCDE
TB = TEXT/ 'C'
RSLT=INDXF(TA,TB)                     $$ RSLT = 3

**********
DASH = TEXT / '-'                     $$ TEXT STRING
T1 = TEXT/ 'THIS IS A - TEST '
DLOC = INDXF(T1,DASH)                 $$ Locate the first dash in T1; DLOC = 11
**********
DL = TEXT / 'DL'                      $$ TEXT STRING
T1 = TEXT/ 'THIS IS A - TEST '
DLOC = INDXF(T1,DL)                   $$ Locate the string DL in T1; DLOC = 0

T1=TEXT/"1234*5678*ABCD*EFGH'         $$ Locate the first and last * in T1
T2=TEXT/'*'
I1=INDXF(T1,T2)                       $$ Find the first * in T1; I1=5
I1-INDXF(T1,T2,1)                     $$ Find the last * in T1; I1=15
```

4.21.7 The SCALF Function

***rslt* = SCALF(*t1*)**

rslt: Scalar value of text string,
t1: Text string to be converted

SCALF converts a text string of numbers to a scalar value, no format is needed. It will return a zero (0) if T1 is a non-number string.

Example:

Get the tool length from the user, turn it into a real number and output it via LOADTL.

```
T1 = TEXT/ 'ENTER TOOL LENGTH:'
RSLT = FILEF(0,1,T1)
T2 = TEXT/READ,0
TLN = SCALF(T2)
LOADTL/1,LENGTH,TLN
```

Optional Format:

Parsing a string with multiple scalars (tokens)

***rslt* = SCALF(*t1,n1,t2,d1*)**

rslt: Zero is returned if a pair is found, one if there is no more tokens.
t1: Text string to be parsed
n1: Desired token number (1st pair is token one, 2nd is token two, etc.)
t2: String where the parsed string is stored.
d1: Scalar where the parsed scalar is stored.

SCALF parses the text string, ***t1,*** based on the token number ***n1***. Then saves the text part of the parsed string in string ***t2*** and saves the numeric part of the string in scalar ***d1***. If there is no text part to the parsed string, then ***t2*** will be set to the text string 'EMPTY' (i.e. T2=TEXT/'EMPTY'). If there is no numeric part to the parsed string, then ***d1*** will be set to 999999 (i.e. D2=999999).

The string ***t1*** is parsed into tokens from left to right. A token ends when the next character read is a comma, blank space or the end of the string or when the previous character is +-0123456789 and the next character is not +-0123456789. A string can contain numerous tokens.

Note*:* Commas and blank spaces are not written to the strings as they are delimiters and they are removed.

Example:

T1 = TEXT/ 'N1 G01 X10.275 Y-3.0 Z-10.5 F10.0'

This string contains 6 tokens and is parsed as follows:

Token	Text	Scalar
1	N	1
2	G	1
3	X	10.275
4	Y	-3.0
5	Z	-10.5
6	F	10.0

Example:

T1 = TEXT/ 'N1 (Austin N.C., Inc.)'

This string contains 6 tokens and is parsed as follows:

Token	Text	Scalar
1	N	1
2	(Austin	99999
3	N	999999
4	C	999999
5	Inc	999999
6	)	999999

Example:

T1 = TEXT/ 'N1 T12 M06 (Tool 12)'

This string contains 6 tokens and is parsed as follows:

Token	Text	Scalar
1	N	1
2	T	12
3	M	6
4	(Tool	999999
5	EMPTY	12
6	)	999999

Example: Parse this string for the 3rd token:

T1 = TEXT/ 'N1 T12 M06 (Tool 12)'
rslt = SCALF(T1,3,T2,D1)

As parsed:

Token	Text	Scalar
1	N	1
2	T	12
3	M	6
4	(Tool	999999
5	EMPTY	12
6	)	999999

rslt = 0

There are 6 tokens so rslt will be set to zero since the command to return the 3rd token was successful.

T2 = TEXT/'M'

T2 is set to the text string of the 3rd token.

D1 = 6

D1 is set to the scalar value of the 3rd token.

Example*:* Parse this string for the 1st 3 tokens:

T1 = TEXT/ 'N1 X10.275'

To parse this string for the 1st token:

rslt = SCALF(T1,1,T2,D1)

As parsed:

Token	Text	Scalar
1	N	1
2	X	10.275

rslt = 0

There are 2 tokens so rslt will be set to zero since the command to return the 1st token was successful.

T2 = TEXT/'N'

T2 is set to the text string of the 1st token.

D1 = 1

D1 is set to the scalar value of the 1st token.

To parse this string for the 2nd token:

rslt = SCALF(T1,2,T3,D2)
rslt = 0

There are 2 tokens so rslt will be set to zero since the command to return the 2nd token was successful.

T3 = TEXT/'X'

T3 is set to the text string of the 2nd token.

D2 = 10.275

D2 is set to the scalar value of the 2nd token.

To parse this string for the 3rd token:

rslt = SCALF(T1,3,T4,D3)
rslt = 1

There are 2 tokens so rslt will be set to one since the command to return the 3rd token was failed.

T4 = TEXT/'EMPTY'

T4 is set to the text string of the 3rd token.

D3 = 999999

D3 is set to the scalar value of the 3rd token.

Example: Using a DO loop, parse the string T1 and store the results in an array:

```
T1 = TEXT/ 'N1 X3.5 Y40.0 Z10.0 F100.00 B30.0 C20.0'
```

As parsed:

Token	Text	Scalar
1	N	1
2	X	3.5
3	Y	40.0
4	Z	10.0
5	F	100.00
6	B	30.0
7	C	20.0

```
RESERV/STR,255,VAL,255
TOKCNT = 0
DO/LP1,NN=1,255,1
 RSLT = SCALF(T1,NN,STR(NN),VAL(NN))
 IF(RSLT.EQ.1)THEN
  NN = 256                          $$ GET OUT OF LOOP
 ELSE
  TOKCNT = TOKCNT + 1               $$ COUNT THE TOKENS
 ENDIF
LP1) CONTIN
MAXTOK = TOKCNT                     $$ SAVE THE NUMBER OF TOKENS
```

Each pass of the DO loop defines a new string (STR) and scalar (VAL).

```
Pass 1, NN=1
        RSLT = 0
        STR(1) = TEXT/'N'
        VAL(1) = 1
        TOKCNT = 1
Pass 2, NN=2
        RSLT = 0
        STR(2) = TEXT/'X'
        VAL(2) = 3.5
        TOKCNT = 2
Pass 3, NN=3
        RSLT = 0
        STR(3) = TEXT/'Y'
        VAL(3) = 40.0
        TOKCNT = 3
Pass 4, NN=4
        RSLT = 0
        STR(4) = TEXT/'Z'
        VAL(4) = 10.0
        TOKCNT = 4
```

```
Pass 5, NN=5
        RSLT = 0
        STR(5) = TEXT/'F'
        VAL(5) = 100.00
        TOKCNT = 5

Pass 6, NN=6
        RSLT = 0
        STR(6) = TEXT/'B'
        VAL(6) = 30.0
        TOKCNT = 6
Pass 7, NN=7
        RSLT = 0
        STR(7) = TEXT/'C'
        VAL(7) = 20.0
        TOKCNT = 7
Pass 8, NN=8
        RSLT = 1
        NN = 256
        MAXTOK = TOKCNT     (or 7)
```

4.21.8 The SPWNF Function

***rslt* = SPWNF(*t1, [t2]*)**

rslt: Zero returned
t1: Text string
t2: 0=process the spawned command at FINI. This will allow you to manipulate the output files from the post processor after they are completed.
1=wait for process to return before continuing. (Default)
2=do not wait for process to return to continue.

SPWNF spawns a sub-process. The ***t1*** text string contains the command line to be passed to the operating system. The FIL run waits until the process is completed.

Example:

```
T1=TEXT/ '$ DIRECTORY'
RSLT=SPWNF(T1)              $$ LIST FILES AT TERMINAL
```

Example:

```
TMP = SPNWF(T1,2)
```

Note:	To execute a BAT file on Windows, you need to run the utility **rbatfile.exe**. This file is in the **\CAMSYS** directory. *** Also see POSTF(34) for advanced usage of SPWNF.

Example:

```
T1 = TEXT/'C:\CAMSYS\RBATFILE.EXE SAMP.BAT'
RSLT = SPNWF(T1)
```

4.22 Text

The **TEXT** statement allows you to assign a symbol to a string of characters. You can subsequently reference the symbol in statements of the type that use character data, such as **PARTNO**, **PPRINT**, and **INSERT**, or in other **TEXT** statements. **TEXT** can be defined as a single string of characters or as a composite string composed of two or more substrings:

symbol*=TEXT/*string
symbol*=TEXT/*string 1, string 2, ---, string n

A string in a text definition can be specified in the following ways:

- As a literal string between quotes.
- By the symbol assigned to a string in a previous text definition.
- As a scalar that is to be converted to a string.

4.22.1 Literal Strings

The format of a literal string is:

'characters'

The string can consist of a variable number of alphabetic, numeric, and/or special characters between two quotes (apostrophes). Blanks count as significant characters of the string. The first character of the string is in the first column following the first quote; the last is in the column immediately preceding the second quote. The string, then, contains as many characters as there are columns between the quotes. The quotes define the limits of the string but are not themselves part of the string.

Example:

($_b$ represents a blank column)

T1 =TEXT/'BEGIN$_b$OPERATION$_b$2'

This statement assigns the symbol **T1** to the specified string, which consists of 17 characters including two blanks. A punctuation character that has a special meaning when not in a string loses the meaning in a string. For example, a dollar sign between quotes is treated as a character of the string, not as an indication of continuation. To continue a literal string onto the next line, you must break it into two strings:

T2=TEXT/'CHANGE$_b$TO$_b$DRILLING', $,'$_b$TOOL'

This statement is equivalent to:

T2=TEXT/'CHANGE$_b$TO$_b$DRILLING$_b$TOOL'

If you wish to include a quote as a character of a string, you must include two quotes; a single quote would be interpreted as the quote that ends the string. When FIL encounters two consecutive quotes following the initial quote that starts a string, it inserts a single quote as a character of the string.

Example:

T3=TEXT/'DON''T$_b$MOVE$_b$PARALLEL$_b$TO$_b$X'

This statement generates the following 24 character string:

DON'T$_b$MOVE$_b$PARALLEL$_b$TO$_b$X

You can reference a previously defined string to cause the characters of that string to be included in the string being defined.

Examples:

TEXT definition Generated string

T1=TEXT/'ABCDEF'	**ABCDEF**
T2=TEXT/'GHI'	**GHI**
T3=TEXT/T1	**ABCDEF**
T4=TEXT/T1, T2	**ABCDEFGHI**
T5=TEXT/'12', T1	**12ABCDEF**
T6=TEXT/'12', T1,'34', T2	**12ABCDEF34GHI**

The maximum number of characters allowed in a string is 128. The minimum number of characters allowed in a string is zero. You can define a null string having no characters by specifying quotes in two consecutive columns. This type of string could be useful in cases where you want to be able to vary whether certain characters appear in a string or not.

Example:

IF(KUL.EQ.0)T1=TEXT/'', ELSE, T1=TEXT/' $_b$AND$_b$COOLNT$_b$OFF'
T2=TEXT/'TURN$_b$SPINDLE$_b$OFF', T1

The string named **T2** varies depending on the value of **KUL** as follows:

KUL= $_b$TURN$_b$SPINDLE$_b$OFF
KUL= $_b$TURN$_b$SPINDLE$_b$OFF$_b$AND$_b$COOLNT$_b$OFF

4.22.2 The REPEAT Modifier

The following sequence causes the characters between the quotes to be repeated n times:

REPEAT, *n*, *'characters'*

Examples:

T1=TEXT/REPEAT, 150,'('

This generates a string composed of 150 left parentheses. (Left parentheses are output by some post processors to represent leader code).

T2=TEXT/REPEAT, 30, '$_b$', 'START$_b$SPINDLE'

This illustrates how **REPEAT** can be used to generate blanks for spacing purposes in a listing.

T3=TEXT/'A', REPEAT, 3, 'BC', 'DEFG'

This generates the following string: **ABCBCBCDEFG**

4.22.3 The MODIFY Modifier

tl* = TEXT/MODIFY,*t2,t3,t4,n1

tl: New modified string
t2: Old string to be modified
t3: Target string
t4: Replacement string
nl: Number of times to replace the target string (use 0 to replace all occurrences)

This enables you to perform a search/replace action on a string. The first example shows how to change a single character.

The second example shows how to replace all occurrences of a space character with an underscore character. In this case, ***t2*** returns the string **'THIS_IS_A_TEST.**

Note: TEXT/Modify allows the replacement string, T4, to be a null string (zero characters, T4=TEXT/''). You can use this to remove sub-strings.

Example.

```
T2 = TEXT/'N001G01X5C-4'          $$ First example
TA = TEXT/'C'
TB = TEXT/'B'
T1 = TEXT/MODIFY,T2,TA,TB,0       $$ Change C to B

T2 = TEXT/'THIS␣IS␣A␣TEST'        $$ Second example
T3 = TEXT/'␣'                     $$ Search for a space
T4 = TEXT/'_'                     $$ Replace with _
T1 = TEXT/MODIFY,T2,T3,T4,0 $$ Find all occurrences
```

4.22.4 The OMIT Modifier

t1* = TEXT/OMIT,*t2,n1

***t1*: New modified string**
***t2*: Old string to be modified**
n1*: Remove blanks in *t2* and return in *tl

***n1:* = 1 Remove trailing blanks**
***n1*: = 2 Remove leading blanks**
***n1*: = 3 Remove all blanks**

Note: If the blank removal creates an entirely blank string, the system will return a string with one blank space.

Example:

In this example, we remove the trailing blanks from the text string.

```
TA = TEXT/'ABCDE bbbbbbb '
TB = TEXT/OMIT,TA,1              $$ TB will be 'ABCDE'
```

Example:

In this example, the string is one blank space, and the modified string will be one blank space.

```
REDEF/ON
T1 = TEST/'                      $$ A STRING CONTAINING ONLY BLANKS
T2 = TEST/OMIT,T1,3              $$ ERASE LEADING AND TRAILING BLANKS
NCH = CANF(52,1)                 $$ NCH=1, AS T2 IS A ONE BLANK SPACE
```

4.22.5 The RANGE Modifier

t1* = TEXT/RANGE,*t2,n1,n2

t1: **New modified string**
***t2*:** **Old string to be modified**
n1: **Starting location in *t2***
n2: **Ending location in *t2***

Substring of ***t2*** returned in ***t1.***

Example:

```
TA = TEXT/'ABCDEFGHIJK'
TB = TEXT/RANGE,TA,3,5       $$ TB will be 'CDE'
```

4.22.6 The READ Modifier

t1* = TEXT/READ,*fn

t1: **Text string for the record read**
fn: **0,1, or 2 (**File number opened with previous **FILEF** function)

The **READ** option in text reads the next record from a file. If **READ** reaches the end of the file, ***t1*** contains the text string **'ERROR$EOF'** to indicate an end of file condition.

If ***fn*** is 0, input is read from the terminal.

4.22.7 The READ,PRINT Modifier

t1 **= TEXT/READ,PRINT**

t1: **Text string for the record read**

The **READ,PRINT** option reads the next record from the auxiliary print file. The auxiliary print file is created using the **POSTF(25,*n*)** command. If **READ,PRINT** reaches the end of the file, ***t1*** contains the text string **'ERROR$EOF'** to indicate an end of file condition.

4.22.8 The READ,PUNCH Modifier

t1 **= TEXT/READ,PUNCH**

t1: **Text string for the record read**

The **READ,PUNCH** option reads the next record from the auxiliary punch file. The auxiliary punch file is created using the **POSTF(25,*n*)** command. If **READ,PUNCH** reaches the end of the file, ***t1*** contains the text string **'ERROR$EOF'** to indicate an end of file condition.

4.22.9 The READ,CHECK Modifier

t1 **= TEXT/READ,CHECK,*n***

t1: **Text string for the record read**

This will return the **"warning or comment"** number "n". If ***n*** is greater than total warnings, it will return **"ERROR$EOF".** To save space, only the first 500 warnings are saved.

This is similar to the **TEXT/READ,PRINT** command.

To save comments, **PLABEL/OPTION,6,TO,2** must be set.

4.22.10 The LAST,3-4 Modifier

The **_MCDWT** (MCD Write) macro is used to edit the current output block just before it is written to the output files. There are two output files written to by the G-Post, the LST file and the MCD file.

To enable the _MCDWT macro, **PLABEL/OPTION,90,TO,2** must be set. This can be set in the Option File Generator on the "General" tab of the "Start/End of Program" panel or it can be set directly in the FIL file by using the command: **PLABEL/OPTION,90,TO,2** to enable and **PLABEL/OPTION,90,TO,0** to disable.

Once set, the **_MCDWT** macro **must** be present somewhere in the FIL file, and will be called every time the G-POST is ready to output a block to the output files.

These two **TEXT** commands, **T1=TEXT/LAST,3 (.LST Block)** and **T2=TEXT/LAST,4 (MCD Block)** are used to get the contents of the output block. See the **TEXT/LAST** section of this manual for more details.

***t1* = TEXT/LAST,3**

t1: **Text string for the current output to the LST file**

The **LAST,3** option reads the current output buffer for the LST file. This is the string that would have been written to the LST file if the _MCDWT macro had not been called by FIL.

***t1* = TEXT/LAST,4**

t1: **Text string for the current output to the MCD file**

The **LAST,4** option reads the current output buffer for the MCD file. This is the string that would have been written to the MCD file if the _MCDWT macro had not been called by FIL.

The **_MCDWT** macro **must** be defined within the current FIL file.

4.22.11 The CONVS Modifier

CONVS enables you to add non-printing characters to a text string.

T1 = TEXT/CONVS,*n*

n Any scalar that specifies the desired ASCII character.

Example.

```
T1 = TEXT/'ABC',CONVS,7      $$ Special char BELL in T1
```

4.22.12 The TIMES Modifier

TIMES[,*n*] generates the current date and time as a string. Using the ***n*** argument you can specify which part of the date and time you desire.

- ***n***=0 Returns the complete date and time. The year is specified with two digits (i.e. 02 for 2002). This is the same result as not specifying the ***n*** argument
- ***n***=1 Returns the complete date and time. The year is specified with four digits (i.e.2002 for 2002).
- ***n***=2 Returns the two digit date only. The time is removed.
- ***n***=3 Returns the time only. The date is removed.
- ***n***=12 Returns the four digit date only. The time is removed.

Example:

```
T1 = TEXT/TIMES       $$ T1 = TEXT/'06/08/02 16:23:00'
T1 = TEXT/TIMES,1     $$ T1 = TEXT/'06/08/2002 16:23:00'
T1 = TEXT/TIMES,2     $$ T1 = TEXT/'06/08/02'
T1 = TEXT/TIMES,3     $$ T1 = TEXT/'16:23:00'
T1 = TEXT/TIMES.12    $$ T1 = TEXT/'06/08/2002'
```

4.22.13 The DATA,*t1* Modifier

DATA,*t1* is used to obtain the current string from an environment variable. ***t1*** specifies the desired environment variable.

If the desired environment variable is not defined the result will be blank.

The result is limited to 128 characters. If the result is greater than 128 characters, it will be trimmed.

The result is case sensitive, a mixed string of upper and lower case characters will be returned. No automatic uppercase conversion will take place.

Example:

Assume your PATH is defined as follows:

Path=C:\;c:\program files\windows nt\accessories;d:\msdev\devstudio\vc\bin;

```
T1=TEXT/'Path'
TPN=TEXT/DATA,T1
$$ TPN=TEXT/ 'C:\;c:\program files\windows nt\accessories;d:\msdev\devstudio\vc\bin;'
```

Example:

Assume your CAMSYS is defined as follows:

CAMSYS=C:\AUSTINNC\CAMSYS

```
T1=TEXT/'CAMSYS'
TPN=TEXT/DATA,T1
$$ TPN=TEXT/ 'C:\AUSTINNC\CAMSYS'
```

4.22.14 The PART[,n] Modifier

PART[,*n*] is used to obtain the current input file name in a string. **PART** returns the complete drive, path, filename and extension. Using the ***n*** argument you can specify which part of the complete filename you desire.

Note: The file name is operating system (UNIX, Windows) dependant.

- ***n***=0 Returns the complete drive, path, filename and extension. This is the same result as not specifying the ***n*** argument
- ***n***=1 Returns the filename and extension only. The drive and path are removed.
- ***n***=2 Returns the filename only. The drive, path and extension removed.
- ***n***=3 Returns the extension only. The drive, path and filename removed.
- ***n***=4 Returns the drive and path only. The filename and extension are removed.

The post processor PGM (program number) is stored in the option file and contains three parts, the Prefix, the Program Number and the Postfix.

- ***n***=11 Returns the PGM program number. The prefix and postfix are removed.
- ***n***=12 Returns the PGM prefix only. The program number and postfix are removed.
- ***n***=13 Returns the PGM postfix only. The prefix and PGM are removed.

Example:

Assume your input file name is **C:\AUSTINNC\TEST.ACL**

```
TPN=TEXT/PART       $$ TPN=TEXT/ 'C:\ AUSTINNC\TEST.ACL'
TPN=TEXT/PART,0     $$ TPN=TEXT/ 'C:\AUSTI\NC\TEST.ACL'
TPN=TEXT/PART,1     $$ TPN=TEXT/ 'TEST.ACL'
TPN=TEXT/PART,2     $$ TPN=TEXT/ 'TEST'
TPN=TEXT/PART,3     $$ TPN=TEXT/ 'ACL'
TPN=TEXT/PART,4     $$ TPN=TEXT/ 'C:\AUSTINNC\'
```

Assume your PGM is **PGM(123456)**

```
TPN=TEXT/PART,11    $$ TPN=TEXT/ '123456'
TPN=TEXT/PART,12    $$ TPN=TEXT/ 'PGM('
TPN=TEXT/PART,13    $$ TPN=TEXT/ ')'
```

4.22.15 The UP Modifier

UP is used to convert all characters in the string to uppercase.

Example:

```
T1 = TEXT/'this is a lower case text string'
T2 = TEXT/UP,T1                          $$ T1 is converted to upper case.
```

4.22.16 The LOW Modifier

LOW is used to convert all characters in the string to lowercase.

Example:

```
T1 = TEXT/'THIS IS AN UPPER CASE TEXT STRING'
T2 = TEXT/LOW,T1                         $$ T1 is converted to lower case.
```

4.22.17 The SIZE Modifier

This command is used to instruct the system the minimum length of TEXT variables. The default length is 80 characters. If you are manipulating TEXT variables with a length greater that the specified SIZE value the system performance will decrease. You will not need to change this value unless you are manipulating large amounts of TEXT strings greater than 80 characters. The maximum value for the minimum length is 128.

Example:

```
T1 = TEXT/SIZE,128                  $$ specifies the minimum length of TEXT variables.
```

4.22.18 Scalars

FIL maintains scalars in a form suitable for computing but not for strings, which means that a scalar must be converted to character form before it can be included in a string. There are many possible character representations for a given scalar, depending on such factors as the number of characters, number of decimal places, etc. For example, the following are three of the many possible representations of the scalar 2.53:

2.53 (4 characters)
$_b$2.53 (5 characters)
$_b$2.530 (6 characters)

If you wish to represent the value of a scalar in a string, you can let FIL convert the scalar using its own format conventions or you can specify explicitly the desired format by using a conversion modifier. When a scalar symbol or literal number without conversion specifications appears in a **TEXT** definition, FIL generates a string that has a format suitable for printing, as in a **PPRINT** statement. A blank is as the first character for spacing purposes. If the scalar is negative, a minus sign is included next. No sign is included for a positive scalar. The numeric characters follow. If the scalar is an integer, no decimal point nor fractional digits are included. If the scalar has a fractional part, a decimal point and as fractional digits are included-but trailing zeros of the fraction are omitted.

Scalar	String	
35	$_b$35	(3 characters)
11.23786	$_b$11.23786	(9 characters)
-725.27	$_b$-725.27	(8 characters)

In the following example, two scalars - tool number ***TN*** and tool length ***TL*** are to be converted to characters and included in a string:

T1=TEXT/'LOAD$_b$TOOL$_b$NO.', *TN*,' $_{bbb}$LENGTH=', *TL*

TN equals 3 and ***TL*** equals 4.75, the following string is generated:

LOAD$_b$TOOL$_b$NO. $_b$3$_{bbb}$LENGTH=$_b$4.75

4.22.19 Conversion Modifiers

The conversion modifiers **CONVI**, **CONVF**, and **CONVE** allow you to specify explicitly the format of the string to which a scalar is to be converted. They can be used in text statements in sequences with the following format:

CONVI
CONVF, *scalar*, *n*, *additional specifications*
CONVE

Scalar specifies the scalar to be converted. It can be a scalar symbol or a literal number. ***n*** specifies the number of characters in the generated string. Its maximum legal value is 20. The additional specifications, if any, vary depending on which modifier is used, as explained in the following sections.

4.22.19.1 CONVI,scalar, n

CONVI (convert to integer) indicates that the specified ***scalar*** is to be converted to a string composed of ***n*** characters. The scalar is truncated before being converted; that is, its fractional part, if any, is deleted. The resulting string represents an integer value. The string is right adjusted; that is, the nth (rightmost) character represents the units digit of the scalar. A minus sign precedes the numeric characters if the original scalar is negative. No sign is included for a positive scalar. If fewer than ***n*** characters are required to represent the scalar, the unused positions to the left are filled with blanks. If more than ***n*** characters are required (including the minus sign for a negative scalar), ***n*** asterisks are generated as an error indication.

Examples:

A = 23
B = -7
C = 125.9

TEXT definition	**Generated string**
T1=TEXT/CONVI,A,2	**23**
T2=TEXT/CONVI,A,3	*b***23**
T3=TEXT/CONVI,A,4	*bb***23**
T4=TEXT/CONVI,B,3	*b***-7**
T5=TEXT/CONVI,B,2	**-7**
T6=TEXT/CONVI,C,4	*b***125**
T7=TEXT/CONVI,C,3	**125**
T8=TEXT/CONVI,C,2	****

Two asterisks are generated for T8 as an error indication because the specified number of characters, 2, is not large enough to represent the scalar 125.

4.22.19.2 CONVF, scalar, n, d

CONVF (convert to fixed point) indicates that the specified scalar is to be rounded to ***d*** decimal places and converted to a string composed of ***n*** characters. The string contains a decimal point followed by ***d*** fractional digits. The string is right adjusted; that is, the nth (rightmost) character of the string represents the dth digit of the fraction. (If ***d*** is 0, the nth character is the decimal point). If the scalar is negative, a minus sign precedes the numeric characters. No sign is included for a positive scalar. If fewer than ***n*** characters are required to represent the scalar, the string is filled to the left with blanks. If more than ***n*** characters are required (including the decimal point and the minus sign for a negative scalar), ***n*** asterisks are generated as an error indication.

Examples:

A = 12.3456
B = -7.654321

TEXT definition	**Generated string**
T1=TEXT/CONVF,A,7,4	**12.3456**
T2=TEXT/CONVF,A,8,4	*b***12.3456**
T3=TEXT/CONVF,A,8,3	*bb***12.346**
T4=TEXT/CONVF,A,10,5	*bb***12.34560**
T5=TEXT/CONVF,B,9,6	**-7.645321**
T6=TEXT/CONVF,B,10,4	*bbb***-7.6453**
T7=TEXT/CONVF,B,6,4	**********

4.22.19.3 CONVF, scalar,n,d,sign option, decimal option,zero option

This expanded format for **CONVF** allows you to specify three options for controlling the inclusion or omission of the sign, decimal point, and leading and trailing zeros. These options could be useful when the exact format of the string is important; for example, when the string is being used with **INSERT** to generate a punched tape block. The permissible values for these options are:

sign option	= 0	Include a minus sign for a negative scalar, no sign for a positive scalar.
	= 1	Always include the sign - plus for a positive scalar, minus for a negative scalar.
decimal option	= 0	Include a decimal point.
	= 1	Don't include a decimal point.
zero option	= 0	Convert leading zeros to blanks.
	= 1	Omit no zeros.
	= 2	Omit trailing zeros.
	= 3	Omit leading zeros and left adjust.
	= 4	Omit leading and trailing zeros and left adjust.
	= 5	Same as option 4 but adds a zero to both sides of the decimal when needed. That is output 0.0 instead of .0.

When the ***zero option*** is 2, 3, 4 or 5 and zeros are omitted, the number of characters in the generated string is less than the specified number by the number of zeros omitted. You must specify all three options or none of them; either three or six entries must follow **CONVF**. If you don't specify the options, the assumed value for each option is zero.

Examples:

A = 12.0
T1 = TEXT/CONVF,A, 8, 4, 0, 0, ZOP

The following shows the strings generated for various values of the variable **ZOP**:

ZOP	String	
0	$_b$12.0000	Blank leading zeros.
1	012.0000	Omit no zeros.
2	012.	Blank trailing zeros.
3	12.0000	Omit leading zeros, left adjust.
4	12.	Omit leading and trailing zeros, left adjust.
5	12.0	Omit leading and trailing zeros, left adjust, output one zero right

The following example shows **CONVF** being used to generate a punched tape block to be output via an **INSERT** statement. The block is to contain:

1. Word address N.

2. Block number. 3 digits, no zeros omitted. FIL symbol is BLKNO.

3. Word address X.

4. X coordinate. 6 digits (4 fractional).
   Omit trailing zeros, FIL symbol is XVAL.

5. Word address Y.

6. Y coordinate. Same format as X. FIL symbol is YVAL.

Table 5- . $ (End of Block code)

T1 = TEXT/'N', CONVF, BLKNO, 3,0,0,1,1,$
'X', CONVF, XVAL, 8, 4, 0,1, 2, $
'Y',CONVF,YVAL,8,4,0,1,2,'$'

For **BLKNO = 23**, **XVAL = 11.2375**, and **YVAL = 4.672**, the following string is generated:

N023X112375Y04672$

4.22.19.4 CONVE, scalar, n, d

CONVE (convert to exponential notation) provides a notation for representing scalars that are too large or small or whose range of values is too great to be represented by **CONVF**. It expresses the ***scalar S*** in terms of a ***fraction F*** and an ***exponent E*** whose relationship is:

S*=*F**(10**E*)**

Specifically, the string generated by **CONVE** consists of the following:

1. An optional minus sign if the scalar is negative. No sign is included for a positive scalar.

2. A decimal point.

3. A fraction equal to or greater than .1 and less than 1 rounded to ***d*** decimal places.

4. A plus or minus sign indicating the sign of the exponent.

Table 6- . A two digit exponent.

The string is right adjusted; that is, the nth (rightmost) character of the string represents the units digit of the exponent. If fewer than ***n*** characters are required, the string is filled to the left with blanks. If more than ***n*** characters are required, asterisks are output as an error indication. The number of characters ***n*** should normally be at least 5 greater than ***d*** to include positions for the sign, decimal point, exponent sign, and exponent.

Examples:

A = .01236
B = -.01236
C = .1236
D = 1.236
E = 12360.4

TEXT definition	**String**
T1=TEXT/CONVE, A, 12, 6	*bb***.123600-01**
T2=TEXT/CONVE, B, 12, 6	*b***-.123600-01**
T3=TEXT/CONVE, C, 11, 4	*bbb***.1236+00**
T4=TEXT/CONVE, D, 11, 4	*bbb***.1236+01**
T5=TEXT/CONVE, E, 11, 6	*b***.123604+05**

4.22.19.5 OMIT, scalar

The OMIT,scalar option is only valid with the INSERT or PPRINT commands. OMIT will remove the leading blank space inserted when converting a scalar to a sting. Once OMIT is given it applies to all scalars in the command.

Examples:

```
A = 1.2
B = 2.5
INSERT/' OUTPUT X',A,' Y',B,' $'
N001 OUTPUT X 1.2 Y 2.5 $
INSERT/' OUTPUT X',OMIT,A,' Y',B,' $'
N001 OUTPUT X1.2 Y2.5 $

INSERT/' OUTPUT X',OMIT,A,' Y',OMIT,B,' $'
N001 OUTPUT X1.2 Y2.5 $
```

4.23 Character Data Statements

The statements that use character data are PARTNO, PPRINT, and INSERT. Two formats are provided for these statements – the fixed field format and the non-fixed field format.

4.23.1 Fixed Field Format

Columns 1 through 6: Major word
Columns 7 through 72: Characters

The fixed position of the major word in columns 1 through 6 eliminates the need for punctuation to separate it from the character data. Similarly, no quotes are required to delimit the character string, which is assumed to consist of the 66 characters in columns 7 through 72. This type of statement cannot be continued since a dollar sign is assumed to be a character of the string, not an indication of continuation.

Example:

Table 7- . 6

PPRINT$_b$CHANGE$_b$TO$_b$DRILLING$_b$TOOL

4.23.2 Non-Fixed Field

Major word/text specification

Any sequence of entries permitted in a **TEXT** statement is equally valid in the text specification of this type of statement. This includes **TEXT** symbols, literal character strings, etc. For example, the following are equivalent:

T1=TEXT/'CHANGE$_b$TO$_b$DRILLING$_b$TOOL'
PPRINT/T1

or

PPRINT/'CHANGE$_b$TO$_b$DRILLING$_b$TOOL'

Note that a statement label can be included in a character data statement with the non-fixed format but not in one with the fixed format.

Example:

ID1) PPRINT/'SWITCH$_b$TO$_b$REAR$_b$TURRET'

FIL determines whether the fixed or non-fixed format is being used as follows: If the major word is not in columns 1 through 6, the non-fixed format is assumed. If the major word is in columns 1 through 6 but the first non-blank column following column 6 contains a slash, the non-fixed format is assumed. The fixed format is assumed only when the major word is in columns 1 through 6 and the first non-blank column following column 6 doesn't contain a slash.

Following are descriptions of the individual character data statements.

4.23.3 PARTNO

The **PARTNO** (part number) statement is usually the first statement of a part program. Typically, it contains data that identifies the part, such as part number and name. The **PARTNO** data is passed on to the post processor, which normally punches it out at the beginning of the punched tape.

Example:

Fixed Field Format

PARTNO$_b$P/N$_b$873261$_b$OP$_b$4$_b$MACHINE$_b$2351

Non-Fixed Field Format

PARTNO/'P/N$_b$873261$_b$OP$_b$4$_b$MACHINE$_b$2351'

4.23.4 PPRINT[text]

The **PPRINT** (post processor print) statement can be used to pass on comments to the post processor. The data in the **PPRINT** statement is normally printed by the post processor as part of its output listing. The **PPRINT** data also appears in the CL print.

The **[*text*]** from column 7 through 72 of the input statement is used.

Examples:

Fixed Field Format

PPRINT$_b$RETURN$_b$TO$_b$HOME$_b$POSITION

Non-Fixed Field Format

PPRINT/'RETURN$_b$TO$_b$HOME$_b$POSITION'

The **OMIT** option has been added to **PPRINT** to remove the leading blank space from the text string when converting a scalar to a string. Once **OMIT** is specified it applies to all scalars in the command.

Input:

XVAL = 10.5
PPRINT/'THE X VALUE IS X',XVAL
PPRINT/'THE X VALUE IS X',OMIT,XVAL

Output:

N001 (THE X VALUE IS X 10.5)
N002 (THE X VALUE IS X10.5)

4.23.5 INSERT[text]

The data in the **INSERT** statement is passed on to the post processor, which normally outputs it directly to punched tape. Thus, **INSERT** allows you to specify explicitly the contents of a portion of the punched tape.

Note: By default the post processor uses a dollar sign ($) to represent the end of block character.
The **[*text*]** from column 7 through 72 of the input statement is punched.

Blanks in the **[*text*]** will be ignored. The **[*text*]** is punched with sequence number and the current **OPSKIP** condition. End of block code is not generated by the post processor and should appear in **[*text*]**, if needed.

This command should be used only when a post processor command is not available for the control information. When used, the post processor does not verify the validity of this block.

Examples:

Fixed Field Format

INSERT N90OG04XO2M83$

Non-Fixed Field Format

INSERT/'N90OG04XO2M83$'

A continuation character can be used to output long tape blocks.

Set **INTCOM(1951)** to the ASCII value of the continuation character.

You can combine **INSERT** commands into long tape blocks as follows:

Assume **INTCOM(1951)** is set to 92 (the \ backslash)

Input:
INSERT/'N001 G01 \$'
INSERT/' X15.0 \$'
INSERT/' Y10.0 \$'
INSERT/' Z100.0 \$'
INSERT/' F10.0 $'

Output:
N001 G01 X15.0 Y10.0 Z100.0 F10.0 $

The **OMIT** option has been added to **INSERT** to remove the leading blank space from the inserted text string when converting a scalar to a string. Once OMIT is specified it applies to all scalars in the command.

Input:
XVAL = 10.5
INSERT/'G00 X',XVAL
INSERT/'G00 X',OMIT,XVAL

Output:
N001 G00 X 10.5
N002 G00 X10.5

4.23.6 DISPLYstring

This command is similar to **PPRINT**. It use it to pass the string to the post processor and to the punch file as an operator message.

This command enables you to pass a text string to the post processor with one command statement:

DISPLY$_b$THIS$_b$IS$_b$A$_b$TEST

With **PPRINT** , you must use three statements to accomplish the same results:

DISPLY/ON
PPRINT$_b$THIS$_b$IS$_b$A$_b$TEST
DISPLY/OFF

DISPLY*string* has fixed-field format. That is, the command occupies columns 1 through 6. The text ***string*** occupies columns 7 through 72, giving you a maximum text string length of 66 characters. Do not use quotes to set off the string.

This means that if you use a space between **DISPLY** and the ***string***, the space is considered to be the first character in the string.

Note:	If you use a slash in column 7, the command is treated as a standard post processor DISPLY statement.

4.24 Repetitive Programming

FIL programming is often repetitive in nature. Frequently, the same group of statements must be executed several times in a FIL file. Of course, statements can be rewritten each time they are required, but this can be a tedious, time consuming task. Also, the number of repetitions may not be known at the time the program is being written but may be dependent on results computed by FIL as the program is executed.

The FIL language provides several features that facilitate repetitive programming, including macros, loops, **IF**, **CASE**, **JUMPTO**, and **DO** statements.

The macro feature allows you to name a group of statements that you can later activate by calling the name.

Looping is the term applied to the process of jumping backward in a FIL program to re-execute a group of statements. Looping can be controlled by **IF**, **CASE**, and **JUMPTO** statements or in a more automatic manner by **DO** statements.

The **IF** and **CASE** statements allows you to program transfers to other statements on a conditional basis. The **JUMPTO** statement allows you to program an unconditional transfer to another statement.

4.24.1 Macros

4.24.1.1 Macro Definition

You can assign a name to a group of statements and subsequently cause them to be executed each time you "call' the name. The group of statements is called a macro. This term is used because the word "macro" literally means "large".

A macro is defined by the following series of statements:

```
macnam = MACRO/varl, var2, -----, varn
        FIL Statements
        .
        .
TERMAC
```

The MACRO statement indicates the beginning of a macro definition, establishes the name, ***macnam***, by which the macro may subsequently be called, and specifies the names of the macro variables, if any. A macro may have no variables or it may have any number of variables up to a system limit. If a macro has no variables, then the slash is optional. The following, for example, are equivalent:

```
MAC1=MACRO
MAC1=MACRO/
```

The **TERMAC** (terminate macro) statement indicates the end of the macro definition. Between the **MACRO** statement and the **TERMAC** statement, you write the regular FIL statements that comprise the body of
the macro.

The statements in the body of a macro are not executed at the time the macro is defined; they are not executed until the macro is called the first time.

You can cause a previously defined macro to be executed by means of a statement with the following format:

CALL/*macnam, var1=value1, var2=value2, --- varn=valuen*

In the above format, macnam is the name assigned the macro in the **MACRO** statement. The equations ***var 1=value 1***, etc. assign values to the macro variables. If the macro has no variables, then, of course, there will be no argument equations.

The construction and use of a macro may best be illustrated by a series of simple examples. Suppose you need to load a new tool and turn the spindle on several times in a FIL subroutine by means of a group of statements like the following:

```
LOADTL/1,ADJUST,1
SPINDL/500,RPM
FEDRAT/10,IPM
```

Rather than rewriting these statements every time you need to repeat them, you could define them as a macro called, say, **TLCHG**:

```
TLCHG = MACRO
        LOADTL/1,ADJUST,1
        SPINDL/500,RPM
        FEDRAT/10,IPM
TERMAC
```

4.24.1.2 Macro Call

Once you have defined this macro, you can cause it to be executed any number of times by calling it:

```
CALL/TLCHG
```

When the macro is called, the effect is the same as if the statements of the body of the macro appeared in place of the call.

The usefulness of macros would be limited, however, if the statements of a macro had to be exactly the same for each execution, so the ability to vary elements within a macro is provided by macro variables. The tool change macro as written above, for example, can only load tool number 1, turn the spindle on to 500 RPMs, and set the feed rate to 10 IPM. In order to load any tool number, set any spindle speed and feed rate, we can make the tool number, spindle speed , and feed rate a macro variable. We arbitrarily use the symbols ***TLN*** for tool number, ***SPD*** for spindle speed, and ***FED*** for the feed rate, for this purpose.

```
TLCHG = MACRO/ TLN, SPD, FED
        LOADTL/ TLN, ADJUST, 1
        SPINDL/ SPD, RPM
        FEDRAT/ FED, IPM
TERMAC
```

Each time we call this macro, we must assign a value to ***TLN, SPD, FED***

```
CALL/TLCHG, TLN=1, SPD=500, FED=10
CALL/TLCHG, TLN=2, SPD=760, FED=24
CALL/TLCHG, TLN=3, SPD=50, FED=12
```

The first call causes tool number 1 to be loaded, the spindle to be set at 500 RPM, and the feed rate to be set at 10 IPM. The second loads tool number 2, sets the spindle speed to 760 RPM, and sets the feed rate to 24 IPM. The third loads tool number 3, sets the spindle speed to 50 RPM, and sets the feed rate to 12 IPM..

The way in which this is accomplished is as follows: At the time the macro is defined, FIL notes that it has three macro variable, ***TLN, SPD,*** and ***FED***. Whenever the macro is called, the values of ***TLN, SPD,*** and ***FED*** must be specified for that particular execution. As FIL executes each statement of the macro, it checks each element to see if it is a macro variable. If so, the current value assigned to the variable replaces the variable itself Thus, when FIL executes the statement **LOADTL/*TLN*,ADJUST,1**, it notes that ***TLN*** is a macro variable, so it substitutes the current value of ***TLN***, which is 1 for the first call, yielding **LOADTL/1,ADJUST,1**. Similarly, the second call produces **LOADTL/2,ADJUST,1** and the third call, **LOADTL/3,ADJUST,1**.

The macro variables ***TLN, SPD,*** and ***FED*** are actually just place markers, indicating where in the macro the actual values assigned to the variables are to be substituted. The name used for a macro variables are unimportant, subject to the usual rules for FIL symbols; the important thing is that the same name be used in the MACRO statement and in the body of the macro wherever the assigned value is to be substituted.

When a macro has more than one variable, you can assign values to them in any order. The following, for example, are equivalent:

```
CALL/TLCHG,  SPD=500, TLN=1, FED=10
CALL/TLCHG, FED=10, TLN=1, SPD=500
CALL/TLCHG, TLN=1, FED=10, SPD=500
```

As another variation of the **TLCHG** macro, we might want to be able to turn the coolant on using the **FLOOD** or **MIST** commands. To do this, we add one more variable – ***KUL*** for the coolant type:

```
TLCHG = MACRO/ TLN, SPD, FED,KUL
        LOADTL/ TLN, ADJUST, 1
        SPINDL/ SPD, RPM
        FEDRAT/ FED, IPM
        COOLNT/KUL
TERMAC
```

When we call this macro, we must assign values to four variables:

```
CALL/TLCHG, TLN=1, FED=10, SPD=500,KUL=FLOOD
```

This call has the same effect as executing the following statements:

```
LOADTL/1, ADJUST, 1
SPINDL/500, RPM
FEDRAT/10, IPM
COOLNT/FLOOD
```

This version of the macro illustrates that macro variables can be used for scalars (***TLN***, ***FED***, and ***SPD***), and vocabulary words (***KUL***). Marco variables can also be used for symbols as follows:

```
TLCHG = MACRO/ TLN, SPD, FED,KUL,HPT
        GOTO/HPT
        LOADTL/ TLN, ADJUST, 1
        SPINDL/ SPD, RPM
        FEDRAT/ FED, IPM
        COOLNT/KUL
TERMAC

P1 = POINT / 10, 10, 10
CALL/TLCHG,TLN=1,FED=10,SPD=500,KUL=FLOOD,$
                HPT=P1
```

4.24.1.3 Normal Values

As an alternative to assigning a value to a macro variable when you call the macro, you can assign a so-called normal value to a macro variable in the **MACRO** statement when you define the macro. When you call the macro subsequently, if you do not assign a value to the variable, it assumes the normal value. If you do assign a value in the call, it overrides the normal value for that call only. If the value is omitted in a subsequent call, it once again assumes the normal value. If a macro has several variables, it is permissible for some of them to have normal values and others not to have them.

Using the tool change macro again as an example, suppose that for most calls we want the feed rate statement to be **FEDRAT/10, IPM**. Rather than repetitively specifying ***FED*=10** in these calls, we assign normal values in the macro statement:

```
TLCHG = MACRO/ TLN, SPD, FED=10, KUL
        LOADTL/ TLN, ADJUST, 1
        SPINDL/ SPD, RPM
        FEDRAT/ FED, IPM
        COOLNT/KUL
TERMAC
```

With this version of the macro, we need to assign a values to ***FED*** in the call statement only when they differ from the normal values:

```
CALL/TLCHG,TLN=1,SPD=500,KUL=FLOOD
CALL/TLCHG,TLN=1,SPD=500,KUL=FLOOD, FED=22
CALL/TLCHG,TLN=1,SPD=500,KUL=FLOOD
```

In the first call, ***FED*** assumes its normal value. In the second call, the normal value is overridden for that call only. In the third call, the normal value is again assumed.

Every macro variable must have a value assigned for every call, either in the CALL statement itself or as a normal value in the MACRO statement.

4.24.1.4 Additional Rules for Using Macros

You cannot define a macro within another macro, but you can call one macro from within another one.

Example :

```
MC1=MACRO/X, Y
  GOTO/X, Y
TERMAC

MC2=MACRO/A
  CALL/MC1, X=A, Y=20
TERMAC

CALL/MC2, A=10
```

The macro **MC1** is called within the macro **MC2**. This example also illustrates how a macro variable can be passed from an outer macro to an inner macro called by the outer one. The call to **MC2** sets ***A*= 10**. The call to **MC1** sets ***X*=A**, or, since ***A* = 10, *X*=10.** Thus, the **GOTO** statement in **MC1** is executed **GOTO/10, 20.**

The assignment of a value to a macro variable, either in a **MACRO** statement or a **CALL** statement can be accomplished only by a simple equation of the form ***variable*=*value***. Computing expressions nests, or subscripts may not be included in a **MACRO** or **CALL** statement. The following, for example, is not permitted:

```
CALL/MAC5, A = C+3
```

The correct method is:

```
AA = C+3
CALL/MAC5, A=AA
```

If you want to use a subscripted symbol in a macro call, you should make the symbol and the subscript two separate macro variables. The following, for example, is not permitted:

```
MAC=MACRO/P
        .
        .
        GOTO/P
        .
TERMAC

CALL/MAC, P=PT(2)
```

A correct method is:

```
MAC=MACRO/P, SUB
        .
        .
        GOTO/P(SUB)
        .
TERMAC

CALL/MAC, P=PT, SUB=2
```

Almost any element within a statement can be a macro variable. However, a statement label cannot be a macro variable.

When an assigned value is substituted for a macro variable in a statement, the resulting statement must have a legal format. A common way in which this rule is violated is illustrated by the following example:

```
MC=MACRO/X
        .
        .
        X = X + I
        GOTO/X, 0
        .
TERMAC

CALL/MC, X=10
```

When the value assigned to ***X***, 10, is substituted for ***X*** in the statement ***X = X + 1***, the result is **10 = 10 + 1**, which is obviously illegal. The problem arises because a macro variable assigned a numerical value appears to the left of the equal sign. A correct way to accomplish the desired result is:

```
MC=MACRO/X
        .
        .
        XX = X + I
        GOTO/XX, 0
        .
TERMAC

CALL/MC, X=10
```

This solves the problem since ***X*** is no longer left of the equal sign.

You must be careful about defining geometry in a macro. Consider the following:

```
PTMAC=MACRO/A, B
        .
        PI=POINT/A, B
        .
TERMAC

CALL/PTMAC, A=3, B=9
CALL/PTMAC, A=7, B=12
```

An error occurs on the second call because **PI** is being doubly defined. You can avoid this problem by using any of the available methods for redefining geometry – **REDEF/ON,** unnamed nested definitions, etc.

A symbol used for a macro variable in one macro can also be used as a macro variable in other macros and as a regular variable outside of macros. The significance of this is that you can use standard macros in a series of part programs without being concerned about conflicts in symbol usage.

```
MAC1=MACRO/S
        .
        SPINDL/S
        .
TERMAC

S = 2.5
CALL/MAC1, S=150
T = S + 3
```

In the above sequence, there are actually two symbols called ***S***. In the macro statement, ***S*** is established as a macro variable and is therefore treated as such wherever it appears in the macro, such as in the **SPINDL** statement. On the other hand, wherever **S** occurs outside the macro it is regarded as a regular scalar variable except in the **CALL** statement, where it is again treated as a macro variable. In the **CALL** statement, the equation ***S*=150** sets the macro variable, producing the statement **SPINDL/150** but does not interfere with the scalar **S** whose value remains **2.5**. Therefore, the final statement **T = S+3** is computed as **T = 2.5 + 3**, giving a result of **5.5.**

4.24.2 Logic Statements

4.24.2.1 The Arithmetic IF

The arithmetic **IF** statement allows control to be transferred on a conditional basis, depending on the value of a scalar. Its format is:

IF (*scalar*) *label 1, label 2, label 3*

In the above format, the parentheses must enclose a scalar or an expression that yields a scalar value. FIL tests the scalar and transfers to the statement designated by *label 1*, *label 2*, or *label 3* depending on whether the scalar is negative, zero, or positive, respectively.

Examples.-

```
IF(A) ST1, ST2, ST3
ST1)    -
        -
ST2)     -
        -
ST3) –
```

In this example, any of three statements could be the first statement executed following the execution of the **IF** statement, depending on the value of **A**.

If **A** is negative, the statement labeled **ST1** is executed next.

IF **A** is zero, the statement labeled **ST2**, is executed next. The statements between the **IF** and **ST2** are skipped.

If **A** is positive, the statement labeled **ST3** is executed next. The statements between the **IF** and **ST3** are skipped.

The statements whose labels are specified in an **IF** statement can either follow the **IF** statement as in this example, causing a jump forward, or precede it, causing a jump backward.

```
IF( X-10.5 ) LB10, LB12, LB12
```

IF **X-10.5** is negative, jump to **LB10**; if zero or positive, jump to **LB12**.

```
IF( 2*F-G ) A1, A2, A1
```

IF **2*F-G** is non-zero, jump to **A1**; if zero, jump to **A2**.

The following simple examples illustrates how an **IF** statement is used to cause the repetitive execution of a series of statements in order to drill holes at the points (4, 0), (6, 0), (8, 0), and (10,O).

```
      X=4                 $$ SET INITIAL X
S1)   GOTO/X,O            $$ GOTO POINT
      GODLTA/O,O,-I       $$ DRILL
      GODLTA/O,O,I        $$ RETRACT
      X=X+2               $$ SET X FOR NEXT PT
      IF(X-10) S1, S1, S2
S2)
```

The **IF** statement sends control back to **S1** as long as **X** is less than or equal to **10**. On the last execution, the statement **X=X+2** sets **X=12**, causing the **IF** statement to transfer to **S2**.

4.24.2.2 The Logical IF

The logical **IF** statement can be used to test a logical expression in order to cause one of two possible actions to be taken. Depending on whether the logical expression is true or false.

A logical **IF** can have either of the following two formats:

IF (*logical expression*) *sub-statement*

If the logical expression is true, the sub-statement is executed; if false, it is not executed .

IF (*logical expression*) *sub-statement 1*, ELSE, *sub-statement 2*

If the logical expression is true, sub-statement 1 is executed and sub-statement 2 is skipped; if false, sub-statement 1 is skipped and sub-statement 2 is executed.

The sub-statements in a above logical **IF** statement can be any FIL statements except the following:

- Another logical **IF**
- **MACRO**
- **TERMAC**
- **CALL**
- **LOOPST**
- **LOOPND**
- **DO**
- **Fixed Field (PARTNO, PPRINT, etc.)**
- **FINI**

A logical **IF** statement can have a label, but the sub-statements cannot.

4.24.2.3 The Logical IF-THEN-ELSE

The logical **IF-THEN-ELSEIF-ELSE** statement can be used to test a logical expression in order to cause different possible actions to be taken depending on whether the logical expression is true or false.

IF (*logical expression*) *THEN*
.
.
sub-statement
sub-statement
sub-statement
.
.
ELSEIF (*logical expression*) *THEN*
.
.
sub-statement
sub-statement
sub-statement
.
.
ELSE
.
.
sub-statement
sub-statement
sub-statement
.
.
ENDIF

If the logical expression is true, then all sub-statements from the **THEN** down to the **ELSEIF** or **ELSE** are executed and all sub-statements from the **ELSEIF** or **ELSE** to the **ENDIF** are skipped; if false, all sub-statement from the **THEN** to the **ELSEIF** or **ELSE** are skipped, if an **ELSEIF** is used and its logical expression is true, then all sub-statements from the **THEN** down to the next **ELSEIF** or **ELSE** are executed and all sub-statements from the next **ELSEIF** or **ELSE** to the **ENDIF** are skipped, if the logical expression is false then all the sub-statements from the **ELSE** to the **ENDIF** are executed.

IF (*logical expression*) *THEN*
.
.
sub-statement
sub-statement
sub-statement
.
.
ENDIF

If the logical expression is true, then all sub-statements from the **THEN** down to the **ENDIF** are executed; if false, all sub-statement from the **THEN** to the **ENDIF** are skipped.

4.24.2.4 Logical Evaluation

A logical expression has the value true or false. The simplest type of logical expression compares the values of two scalars, as follows:

s1 .op. s2

In the above format, ***sl*** and ***s2*** are scalars-numeric, symbolic, or computing expressions yielding scalar values, and ***.op***. is one of the following relational operators:

.LT.	less than
.LE.	less than or equal to
.EQ.	equal to
.NE.	not equal to
.GE.	greater than or equal to
.GT.	greater than

Note that the first and last characters of logical operators are decimal points. This notation is used because it permits words with the same spellings as operators but without the decimal points to be used as symbols or synonyms. For example, **.LT.** means "less than', but **LT** is still available for use as a symbol or synonym.

Logical operators are handled by FIL similarly to the way arithmetic operators like + (plus) and –(minus) are handled, not like vocabulary words. Therefore, synonyms cannot be used for them, nor may they be assigned to macro variables.

A = 1
B = 4
IF(A .LT. B) JUMPTO/S10

Since **A** is less than **B**, the logical expression is true, so **JUMPTO/S1O** is executed.

IF(A.EQ. B)JUMPTO/S1O

The expression is false, so the **JUMPTO** is not executed.

IF(A+10.LE. B+2)C=3, ELSE, C=4

The expression is false, so **C=4** is executed.

IF(A.GE. 1)FEDRAT/10, ELSE, FEDRAT/20

The expression is true, so **FEDRAT/10** is executed.

IF(A.GT. –B)GOTO/P1, ELSE, GOTO/P2

The expression is true, so **GOTO/P1** is executed.

When programming a logical **IF**, you should remember what constitutes a statement, since a sub-statement can consist of a single statement only. A semicolon always indicates the end of a statement, as does a comma after a single word statement.

Example:

IF(R.GT. 1.5)RAPID; GOTO/P1, ELSE, GOTO/P2 $
IF(R.GT. $1.5)RAPID, GOTO/P1, ELSE, GOTO/P2

The above two statements are incorrect because **RAPID** and **GOTO** are two separate statements and hence cannot be used together as a single sub-statement.

Scalar comparisons can be combined into more complicated logical expressions, as follows:

sc1 .op1. sc2 .op2. sc3, etc.

In the above format, ***sc1, sc2***, ***etc***. are scalar comparisons, and ***op1, op2, etc.*** are logical operators. Each logical operator must be one of the following:

.AND.
.OR.

The expression ***scl* .AND. *sc2*** is true if the first scalar comparison is true and the second is true; if either is false, the expression is false.

The expression ***sc1* .OR. *sc2*** is true if either scalar comparison is true or if both are true; it is false if both are false.

One additional logical operator is **.NOT.** , which can be included at the beginning of **a** logical expression or .following **.AND.** or **.OR.** within a logical expression to reverse the value of the part of the expression to which it applies from true to false or false to true.

Parentheses may be used to designate the order in which logical operations **are** performed. When parentheses are not used, **.NOT.** operations are performed **first,** then **.AND.**, and, finally, **.OR.**.

Example.

```
A = 2
B =.5
C =10
R = .03125

IF( A.GT. 0 .OR.  B.GE. 2 .AND. C.LE. 4) $
   CUTTER/2*R, ELSE, CUTTER/2*R+.1
```

The logical expression is evaluated as follows:

```
A.GT. 0 .OR. (B.GE. 2 .AND. C.LE. 4) = True
.OR. ( False.AND. False) = True
.OR.  False = True
```

The following logical expression has the same format as the one above except that parentheses are used to cause the **.OR.** operation to be performed before the **.AND.**:

```
IF((A .GT. 0 .OR.  B .GE. 2) AND.  C .LE. 4) $
   CUTTER/2*R, ELSE, CUTTER/2*R+.1
```

In this case, the expression is evaluated as follows:

```
(True .OR. False) .AND. False
= True              .AND. False
= False
```

The following example illustrates the use of **.NOT.**

```
X = 1
Y = 7
IF(X GT. 0 .AND.  .NOT.(Y LT. 0 .AND. Y .GT.10))$
  SPINDL/100,RPM, ELSE, SPINDL/5600,SFM
```

The logical expression is evaluated as follows:

```
= True .AND.   .NOT. (False .OR. False)
= True .AND.   .NOT. False
= True .AND.   True
= True
```

4.24.2.5 CONTRL/TOLER,IF,*t*

For the logical **IF/command**, a 'bit by bit' comparison is done in FIL. This may cause two numbers that are nearly the same to fail the comparison test. Prior to **CONTRL/TOLER,IF**,***t*** additional programming was require, something like **IF((ABSF(A-B)) .LT. EPS)** etc.

To avoid the extra coding, **CONTRL/TOLER,IF,*t*** specifies a tolerance '***t***' for **IF/test**, for which A and B will be considered the same. The default value for ***t***=0.

A sample program with CONTRL/TOLER,IF,t is shown below. If you do not use the TOLER command, only line-6 will pass the test and all others will fail. For practical purpose, the numbers A1 and B1 are almost the same.

***Input*:**

```
CONTRL/TOLER,IF,0.00001
A1=1.0
B1=1.0
IF(A1 .EQ. B1)PPRINT/'A1=B1=1.0'
B1=1.0001
IF(A1 .EQ. B1)PPRINT/'A1=B1=1.0001'
B1=1.00005
IF(A1 .EQ. B1)PPRINT/'A1=B1=1.00005'
B1=1.00001
IF(A1 .EQ. B1)PPRINT/'A1=B1=1.00001'
B1=1.000001
IF(A1 .EQ. B1)PPRINT/'A1=B1=1.000001'
B1=1.000002
IF(A1 .EQ. B1)PPRINT/'A1=B1=1.000002'
```

Output:

```
PARTNO  TEST
PPRINT A1=B1=1.0
PPRINT A1=B1=1.000001
PPRINT A1=B1=1.000002
```

4.24.2.6 The CASE Statement

The **CASE** statement enables you to do structured programming. However, it is more versatile than the logical **IF** statements. Since the **CASE** statement is faster and more powerful, we suggest you use it when you need to accommodate more than two situations.

CASE/*s1-t1*
  WHEN/*scalar-t1*
    statement-1
    statement-n
  WHEN/*scalar*,THRU,*scalar*
    statement-1
    statement-n
  WHEN/OTHERS
    statement-1
    statement-n
ENDCAS

s1: Scalar variable to be matched
t1: Text variable to be matched

You can combine situations to include or excluded certain conditions. For example, the following line looks for scalars 2,6,7,8,9,10, and 20

WHEN/2,6,THRU,10,20

When those scalars are encountered, the statements that follow the **WHEN** statement are executed.

WHEN/OTHERS executes the block if a match failed otherwise. It must be the last **WHEN** option in a **CASE** block. **ENDCAS** terminates the **CASE** block.

If you are using a text variable, you must use one of the following formats:

Define the variable first as in the following:

```
T1 = TEXT/'MIST'
CASE/T1
        WHEN/T1
```

You can also define the text within the **WHEN** statement:

```
        WHEN/(TEXT/'MIST')
```

The following is a general example of a **CASE** statement:

```
CASE/s1                                 $$ Set case variable s1
        WHEN/I                          $$ Execute if s1 = 1
                statement
                statement
        WHEN/i1,i2                      $$ Execute if s1 = i1 or i2
                statement
                statement
        WHEN/2,THRU,12                  $$ Execute if s1 is 2 through I2
                statement
                statement
        WHEN/OTHERS                     $$ Execute if s1 is not 1
                statement               $$ or i1 or I2 or 2 through 12
                statement
ENDCAS                                  $$ End of CASE statement

CASE/t1                                 $$ Set case variable t1
        WHEN/(TEXT/'CLW')               $$ Execute if t1=CLW
                statement
                statement
        WHEN/T2,T3                      $$ Execute it t1 = T2 or T3
                statement
                statement
ENDCAS                                  $$ End of CASE statement
```

4.24.2.7 DO Loops

The DO loop feature provides an automatic mechanism for controlling the repetitive execution of a series of statements. A DO loop has the following format:

DO/*label*, *name=first, limit, incr*
        .
***label*)**

The statements starting with the one following the **DO** statement and ending with the labeled statement are executed repetitively as specified by the entries in the **DO** statement, which have the following significance:

label is the label of the final statement of the **DO** loop. The final statement of a **DO** loop cannot be a macro call.

- ***name*** is the symbol for a scalar called the **DO** variable, which is varied for each repetition of the loop.

- ***first, limit***, and ***incr*** are scalars – numeric, symbolic, or nested computations. They control the values of the **DO** variable and the number of times the loop is executed.

- ***first*** is the value of the **DO** variable for the **first** execution of the loop. It may be positive, negative, or zero.

- ***limit*** is the limiting value of the **DO** variable. It may be positive, negative, or zero.

- ***incr*** is the incremental value added to the **DO** variable for each repetition of the loop. It may be positive or negative but not zero. If omitted, incr is assumed to be 1.

For positive ***incr***, stepping of the **DO** variable is from the first value up to the ***limit*** value. If ***incr*** is such that the limit cannot be reached exactly, the final value is the greatest value that can be reached that is less than the limit.

For negative ***incr***, stepping of the **DO** variable is from the first value down to the ***limit*** value. If the limit cannot be reached exactly, the final value is the smallest value that can be reached that is greater than the

For example, the following **DO** statements generate the indicated series of values for the **DO** variable.

	Values of *N*
DO/S1, *N*=1, 4, 1	**(1, 2, 3, 4)**
DO/S1, *N*= 1, 4	**(1, 2, 3, 4)**
DO/S1, *N*= 2, 10, 2	**(2, 4, 6, 8, 10)**
DO/S1, *N*=1, 10, 2	**(1, 3, 5, 7, 9)**
DO/S1, *N*=1.5, 4,.5	**(1.5, 2, 2.5, 3, 3.5, 4)**
DO/S1, *N*=4, 1, -1	**(4, 3, 2, 1)**
DO/S1, *N*=10, 2, -2	**(10, 8, 6, 4, 2)**
DO/S1, *N*=9, 2, -2	**(9, 7, 5, 3)**
DO/S1, *N*=-2, 4, 2	**(-2, 0, 2, 4)**

If the first value equals or is beyond the limit, the loop is executed once, with the **DO** variable equal to the first value. The following statements, for example, cause the loop to be executed once with the indicated value for the **DO** variable:

DO/S1, *N*=2, 2, 1	**(2)**
DO/S1, *N*=4, 1, 1	**(4)**
DO/S1, *N*=1, 4, -1	**(1)**

In addition to serving as a loop counter, the **DO** variable can be used like any scalar variable within the loop - - as a subscript or in computing expressions. The **DO** variable can even be modified within the loop, but this should be done very carefully, if at all, since doing it improperly could cause an endless loop.

When all repetitions of a **DO** loop have been completed, the **DO** variable will retain the value that it had for the final repetition and may subsequently be used outside the loop like a regular scalar variable.

Example:

```
DO/S10, N=1, 6
-
S10)
A=N+1                    (A=6+1=7)
```

A **DO** loop included within another **DO** loop is called a nested **DO** loop. As many as 10 levels of **DO** loops may be in effect at the same time in this manner. If a macro called within a **DO** loop contains a **DO** loop itself, this constitutes a nested **DO** loop.

The following example illustrates one **DO** loop nested within another:

```
DO/S20, L=1, 4
   DO/S1O, N=1, 3
     ---
   S10)---
   ---
S20)CONTIN
```

The outer loop is executed 4 times. The inner loop is executed 3 times for each execution of the outer loop, or 12 times in all.

More than one **DO** loop can end on the same statement, as in the following example:

```
DO/S10, M=1, 3
   DO/S10, N=1, 2
   ---
S10)CONTIN
```

It is not permissible for **DO** loops to overlap as in the following example:

```
DO/S10, M=1, 3
   DO/S20, N=1, 2
       ---
S10)---
  ---
  S20)CONTIN
```

The following example illustrates the use of a nested **DO** loop to generate a rectangular grid of points. For comparison, the example is also programmed using an FIL loop.

Example:

```
DO/S1, Y=1.5, 7.5, 2
 DO/S1, X=2.5, 8.5, 1.5
        GOTO/X, Y, 0
        GODLTA/0, 0, -1
S1) GODLTA/0, 0, 1
        Y = 1.5
S2)     GOTO/X, Y, 0
        GODLTA/0, 0, -1
        GODLTA/0, 0, 1
```

Example: (Continued…)

```
S1)     X=2.5
        X=X+1.5
        IF(X-8.5)S2, S2, S3
S3)     Y=Y+2
        IF(Y-7.5)S1, S1, S4
S4)     ---
```

It is permissible to transfer out of a **DO** loop using an **IF** or **JUMPTO** statement. When this occurs, the **DO** variable retains the value that it had at the time of the transfer.

Example.

```
RESERV/P,50
DO/S1,N=1,50,1
  P(N) = POINT/(N*.5),0,0
S1)
DO/S2,N=1,50,1
  T=CANF(P(N),1)  $$ T = X VALUE OF POINT
  IF(T.GT.10)THEN
    JUMPTO/S3
  ELSE
    GOTO/P(N)
  ENDIF
S2) CONTIN
S3) CONTIN
```

In this particular case, when the x value of the point exceeds 10 the **IF** statement is true resulting in a transfer out of the **DO** loop to **S3**. At this time, **N** is set to the current counter value.

DO-WHEN Loop

This is a special form of a **DO** loop. It executes the loop for the specified **WHEN** values of the **DO** loop variable. In some cases it may be more convenient to use than the regular **DO** loop form.

```
DO/label,name=val1,val2,val3,…,valn,WHEN
…
…
label)CONTIN
```

Example: DO-WHEN loop will process (I1=1, 14, 31, 44, 47).

```
DO/10,I1=1,14,31,44,47,WHEN
    FEDRAT/I1
    GOTO/P1
    …
    …
10)CONTIN
```

Compare the above **DO-WHEN** loop with the following regular **DO**-Loop:

Example: (Continued…)

```
DO/10,I1=1,,50
   IF(I1 .EQ, 1)JUMPTO/5
   IF(I1 .EQ, 14)JUMPTO/5
   IF(I1 .EQ, 31)JUMPTO/5
   IF(I1 .EQ, 44)JUMPTO/5
   IF(I1 .EQ, 47)JUMPTO/5
   JUMPTO/10
   5)CONTIN
   FEDRAT/I1
   GOTO/P1
10)CONTIN
```

4.24.2.8 CONTIN

The **CONTIN** (continue) statement causes no action to be taken by FIL. Its purpose is to provide a dummy statement to which a label can be attached.

Its primary use is as the last statement of a **DO** loop to which an **IF** or **JUMPTO** statement can transfer when it is desired to skip the remaining statements in the loop and cause repetition of the loop to take place.

4.24.2.9 JUMPTO

The **JUMPTO** statement can be used to program an unconditional jump. Its format is:

JUMPTO/*label*

FIL transfers to the statement with the designated ***label*** and executes it next.

For example, suppose that ***F*** is a scalar that represents a feed rate. If ***F*** is 10 or less, set the spindle speed to 200; if greater than 10, set the speed to 250.

```
IF( F-10 ) S1, S1, S2
S1)     SPINDL/200
        JUMPTO/S3
S2)     SPINDL/250
S3)
```

The **JUMPTO** statement is necessary for skipping the statement labeled **S2** when the **IF** transfers to **S1.**

4.24.2.9.1 The Multiple JUMPTO Statement

The multiple **JUMPTO** statement causes a transfer to one of a series of designated statements, depending on the value of a scalar. Its format is:

JUMPTO/*scalar, label 1, label 2, ---, label n*

The scalar is a numeric or symbolic scalar or computing expression. If not an integer, it is truncated to an integer before being used. Following the scalar, a variable number of statement labels may be given.

If the scalar has a value of 1, a transfer is made to the statement with the first designated label; if 2, to the second, etc. An error results if the scalar is negative, zero, or greater than the number of labels.

Example:

```
M=MACRO/N
        JUMPTO/N, S1, S3, S8, S9

        S1)     -
                -
        S3)     -
                -
        S8)     -
                -
        S9)     -
                -
TERMAC

CALL/M, N=1             $$ JUMPTO S1
CALL/M,N=3              $$ JUMPTO S8
CALL/M,N=5              $$ ERROR - - NO 5th LABEL
```

4.24.3 Preprocessed Macros

Note:	Preprocessed Macros are a more complex choice and it is recommended that INCLUDE/BINARY be used as the method for encrypting and protecting the FIL files. See the INCLUDE/BINARY section of this manual.

It is a normal practice to create general purpose macros, such as a "tool change macro" to facilitate part programming. Often, these macros are large in size and are processed at the expense of execution time for each part program. You can use the alternate method to "save" these macros in a translated format (namely pre-processed or system macros) and eliminate re-processing for each part program that uses these macros.

You can activate this option by the commands:

```
PUNCH/20,ALL,filman
PUNCH/20,ALL

READ/20,ALL,filnam
READ/20,macnam
```

"filnam" is a disk file name which contains more than one macro and "macnam" is a disk file with the FIL macro name. Both names must be legal FIL symbols with six or less characters in length and they cannot be FIL vocabulary words.

4.24.3.1 Saving Macros

Prior to utilizing a system macro, you need to create a disk file, which contains the macros in FIL translated format. This is done via a separate FIL execution, in which the part program is error free and contains the statement **PUNCH/20** with the required macros. The only purpose of this run is to create system macros – otherwise it is a dummy run.

If you plan to use a set of macros in all part programs, you would create the file as follows:

```
$$ SAVE MACROS DRL,TCH
PUNCH/20,ALL,EG1
DRL=MACRO/P                $$ DRL
        GOTO/P
        GODLTA/-1
        GODLTA/+1
TERMAC
$$ TCH
TCH=MACRO/T
        LEADER/10
        LOADTL/T
TERMAC
FINI
```

When you execute this part program, a disk file by the name **"EG1"** will be crated with the macros **DRL** and **TCH** in translated format.

If you plan to use the macros **DRL** and **TCH** selectively in the part programs, you can use the **PUNCH/20,ALL** command without specifying a "filnam" as follows:

```
$$ SAVE DRL AND TCH SEPARATELY
PUNCH/20,ALL
DRL=MACRO/P                $$ DRL
        GOTO/P
        GODLTA/-1
        GODLTA/+1
TERMAC
$$ TCH
TCH=MACRO/T
        LEADER/10
        LOADTL/T
TERMAC
FINI
```

When you execute this program, two disk files by the names **DRL and TCH** will be created for the macros as separate files.

When creating system macros, the **PUNCH** command is the first statement and is the only one required. Also, the part program can contain only **MACRO-TERMAC** (with FIL statements in between), $$ comment lines, **SYN, PPWORD, PRINT/IN** and **PRINT/ON-OFF,IN** statements. If any other statement is encountered, the job will be terminated with an error diagnostic. Also, the statements **PUNCH/20 or READ/20** are not allowed inside the **MACRO-TERMAC** pair.

4.24.3.2 Reading Macros

Once you have created the system macro files, you can access them from any part program by the command **READ/20.**

When you want to access the macros created by the first **PUNCH/20,ALL,EG1** example, you would program as follows:

```
$$ READ BOTH DRL AND TCH
READ/20, ALL,EG1
SYN/ON
P1=P5/1,1
P2=PT/2,2
CALL/TCH,T=1
CALL/DRL,P=P1
CALL/DRL,P=P2
FINI
```

When you execute this program, the system macro file **"EG1"** will be read and the macros **DRL** and **TCH** will then be defined.

If you want to access only the macro **DRL,** created by the second **PUNCH/20,ALL** example, you would program as follows:

```
$$ READ ONLY DRL MACRO
READ/20,DRL
SYN/ON
P1=PT/1,1
P2=PT/2,2
CALL/DRL,P=P1
CALL/DRL,P=P2
FINIA
```

When you execute this program, the system macro file "DRL" will be read to define the macro DRL.

4.24.3.3 System Macro Notes

A system macro is processed only once during the creation of the disk file. When it is read into a part program, it is already defined. This is unlike an INCLUD command, where the macro is processed in each part program.

You normally would create a system macro, only when the macro is error free and is logically correct and when it is ready for production. During development of a general purpose macro, a system macro creation is not recommended as no time saving is possible.

The commands SYN,PPWORD,PRINT/IN and PRINT/ON-OFF,IN are pass one commands in FIL and they cannot be saved in a system macro – they have meaning only during a system macro creation. To understand this concept, let us look at the following example:

Example:

```
$$ SYN IN SYSTEM MACRO
PUNCH/20, ALL
MYMAC=MACRO/
        SYN/P,POINT
        SYN/L,LINE
        PO=P/0,0
        LX=L/XAXIS
        LY=L/YAXIS
TERMAC
FINI

$$ READ THE ABOVE MACRO
READ/20,MYMAC
P1=P/1,1
L1=L/1,1,2,2
FINI
```

The second part program will fail, because P and L are unknown vocabulary words. To correct this, place the SYN commands after the READ statement as the SYN commands are not output to the system macro file.

Whenever possible, group a set of macros into categories, such as a set for LATHE, MILL etc. and use the ALL option with a “filnam” to create the system macros. This will keep the number of disk files to a minimum. There is no restriction for the number of READ/20 commands in a part program and can be a combination of one or many macros as long as they are the first statements in a part program. Also reading the file which contains more than one macro first, will reduce execution time.

```
$$ READING MANY MACROS
READ/20,ALL,FILE1
READ/20,MAC1
READ/20,MAC2
READ/20,ALL,FILE2
```

You can define additional macros in a part program along with the system macros as long as the names are not duplicated as shown in the following example:

Example:

```
READ/20,ALL,EG1
PARTNO TEST
CLPRNT
SYN/ON
M1=MACRO/
--
--
TER,MAC
CALL/TCH,T=1          $$ SYSTEM MACRO  CALLED
CALL/M1               $$ LOCAL   MACRO  CALLED
FINI
```

Avoid using FIL vocabulary words as variable symbols within a system macro. For example, the statement T1=CANF(TOOL,1) inside a system macro contains the vocabulary word TOOL. At the time of the macro creation, TOOL is assumed to be a vocabulary word unless it is defined inside the macro – such as TOOL=DATA/1,2,3.

4.24.3.4 PRINT/IN,number

When a system macro is included in a part program, the input line numbers of the statements inside the macro may conflict with the line numbers of the part program. Since error messages always point to line numbers, tracing the errors may become difficult. To eliminate this problem, the new command PRINT/IN, number is provided to specify a starting line number "number" for the part program.

Unique line numbers can be assigned to statements of a system macro during its creation as follows:

```
$$ MACRO STARTS WITH LINE NUMBER 1000
PUNCH/20,ALL
PRINT/IN,1000
MYMAC=MACRO/
--
--
TERMAC
FINI


$$ DEFAULT LINE NUMBER IS 1
$$ FOR THIS PART PROGRAM
READ/20,MYMAC
PARTNO TEST
--
CALL/M1
--
FINI

$$ STARTING LINE NUMBER IS 2000
$$ FOR THIS PART PROGRAM
PRINT/IN,2000
READ/20,MYMAC
PARTNO TEST
--
CALL/M1
--
FINI
```

4.24.4 Encrypting FIL – INCLUD/BINARY

Note:	Using this command is recommended over the Preprocessed Macros procedures.

INCLUD/BINARY,filename

"filename" is the encrypted binary file created by the utility Wncrypt.exe. The purpose is to protect or lock the FIL source data from modification.

This encryption method is much simpler than the preprocessed macros method of PUNCH-READ/20. The encrypted file can be any valid FIL text unlike the preprocessed macros. You can continue to use both methods.

Caution: Once a FIL file is encrypted using Wncrypt.exe it cannot be decrypted. So save your original FIL source files.

On Windows systems, you encrypt a FIL file by executing **\CAMSYS\Wncrypt.exe**

On UNIX systems, you encrypt a FIL file by executing **/unc/camsys/ncrypt.exe**

The encrypt executable will prompt for the input/output file names or you can pass them as arguments as, "Wncrypt.exe Test1.dat Test1.bin", to encrypt the source file Test1.dat into Test1.bin. We suggest you name the output file as *.bin so you can identify them as encrypted files.

When G-Post finds the command **INCLUD/BINARY,filename,** it will look in the local directory and then search the \CAMLIB\ for the file.

- Filename is any legal name and is not restricted to 6 characters.
- INCLUD/BINARY must start in column 1 of the FIL file. Do not space or tab over.
- There is no limit to the number of INCLUD/BINARY statements.
- INCLUD/BINARY can be anywhere in the FIL file as it is treaded like any other FIL input except it has been encrypted.

Example:

1. Create an encrypted file named cm01.dat. This file is to include the data of the MACHIN FIL section

   File cm01.dat:

```
CIMFIL/ON,MACHIN
  MCH = POSTF(7,5)
  DMY = POSTF(13)
CIMFIL/OFF
```

2. Encrypt the cm01.dat file

   Wncrypt.exe cm01.dat cm01.bin

Example: (Continued…)

3. Copy the encrypted file to your \CAMLIB directory

```
Copy  cm01.bin   \CAMLIB\cm01.bin
```

4. Using the encrypted file in your FIL file, UNCX01.F01

File UNCX01.F01:

```
$$ SAMPLE FIL FILE
CIMFIL/ON,PARTNO
     PNTXT = TEXT/CLW
     DMY = POSTF(13)
CIMFIL/OFF

INCLUD/BINARY,CM01.BIN

CIMFIL/ON,SPINDL
     SPED = POSTF(7,4)
     DMY = POSTF(13)
CIMFIL/OFF
FINI
```

5 POSTF Functions

Introduction

This chapter explains each of the **POSTF** function types in numerical order for easy reference. Where feasible, we've included a short example to help explain the function. Be sure to look over the examples in Appendix A as well.

> **Note:** In the syntax description, we use the variable ***rslt***. You can use any variable you want, but be cautious about this. In the instance where zero is returned, be sure to use a variable that you will not be using elsewhere. Remember to use **redef/on** if you deliberately want to redefine text variables.

The syntax to use this function is as follows:

rslt **=POSTF(*function_type*, *arg1*, *arg2*, *arg3*, *argn*)**

rslt is a scalar that reports the result of the operation, with one exception. When you use a **TEXT** function type, ***rslt*** is a **TEXT** string.

Function_type is a two-digit number that represents the function type. The number of arguments needed depends on the type of function.

Table 5-1 shows a list of functions in order by function type number. Table 5-2 shows the same list of functions, grouped by functionality. The last two pages of this chapter contain a **POSTF** function chart that you can copy for handy reference.

Function Type	Functionality
01	Get COMMON value
02	Set COMMON value
03	Set COMMON to empty
04	Test for COMMON empty
05	Get number of words in current CL record
06	Find word type in current CL record
07	Get CL record value
08	Get text data from CL record
09	Put minor word in CL record
10	Put scalar in CL record
11	Reserved for future use
12	Set number words in CL record
13	Process current CL record
14	Read next current CL record from CL file
15	Position to a CL record in CL file
16, 17, 18	Reserved for future use
19	Output current Post block (call out)
20	Save current CL record
21	Load a saved CL record
22	Get current machine number
23	Move COMMON values
24	Turn on/off ISN trace print
25	Redirect Post output<$IPOSTF chart>
26	Control CIMFIL/ON-0FF
27	Security Information
28	Locate a word/scalar/couplet in the current CL record
29	Remove a word/scalar/couplet in the current CL record
30	Read the next CL record of a specified type
31	_OUTPT Macro functions
32	Store/Retrieve scalars from large memory arrays
33	Store/Retrieve text strings from large memory arrays
34	Put G-Post to sleep for a number of seconds

Table 5-1. Function Types By Number

Functionality	Type
Get CL Information	
Get number of words in current CL record	**05**
Find word type in current CL record	**06**
Get CL word record value	**07**
Get text data from current CL record	**08**
Locate a word/scalar/couplet in the current CL record	**28**
Remove a word/scalar/couplet in the current CL record	**29**
Load CL information	
Put minor word in CL record	**09**
Put scalar in CL record	**10**
Set number words in CL record	**12**
Process current CL record	**13**
Save current CL record	**20**
Load a saved CL record	**21**
Manipulate Post COMMON	
Get COMMON value	**01**
Set COMMON value	**02**
Set COMMON to empty	**03**
Test for COMMON empty	**04**
Output current Post block (call out	**19**
Get current machine number	**22**
Move COMMON values	**23**
Redirect Post output	**25**
CL File Positioning	
Read next current CL record from CL file	**14**
Position to a CL record in CL file	**15**
Control CIMFIL/ON-OFF	**26**
Read next CL of the specified type from the CL file	**30**
Debug	
Turn on/off ISN trace print	**24**
Security Information	**27**
Put the G-Post to sleep for a number of seconds	**34**
_OUTPT Macro	
_OUTPT Macro Functions	**31**
Store/Retrieve data from large memory arrays (like RESERV/cmd)	
Store/Retrieve text strings from large memory arrays	**32**
Store/Retrieve text strings from large memory arrays	**33**

Table 5-2. Function Types By Functionality

5.1 Function Type 01 (Get COMMON Value)

***rslt* =POSTF(01,*arg1*,*arg2*)**

rslt: COMMON value returned
arg1: type of COMMON
1= integer
2= real
3= double
arg2: COMMON location number
INTCOM number range is 1 – 3500
RELCOM number range is 1 – 902
DBLCOM number range is 1 – 1000

Use this function type to get the COMMON value. In this example, we want the value of **TULNUM**. We know **TULNUM** is a DBLCOM, so we use 3 for ***arg1***. We use 497, the DBLCOM location, for ***arg2***.

Refer to the appropriate G-Post User's Guide for a description of COMMON locations and contents.

```
TULNUM=POSTF(1,3,497)
```

The above line would be read as follows:

Get the value from DBLCOM 497 and set the variable **TULNUM** equal to that value.

5.2 Function Type 02 (Set COMMON Value)

***rslt* =POSTF(02,*arg1*,*arg2*,*arg3*)**

rslt: zero returned
arg1: type of COMMON
1= integer
2= real
3= double
arg2: COMMON location number to be reset
INTCOM number range is 1 – 3500
RELCOM number range is 1 – 902
DBLCOM number range is 1 – 1000
arg3: Value to be stored in the COMMON location
INTCOM value range is ± 32766
RELCOM value range is ± 99999.99999
DBLCOM value range is ± 99999.99999

Use Function Type 02 to set or reset the COMMON value. For example, you can use Function Type 01 to get the COMMON value, manipulate it the way you want, then use Function Type 02 to write the new value back in. Refer to the appropriate G-Post User's Guide for a description of COMMON locations and contents.

In the preceding example, we used Function Type 01 to determine the value for **TULNUM**. We want to increment that value by 1, then use Function Type 02 to write that new value back into the COMMON.

```
TULNUM = TULNUM + 1          $$ Increment the value by 1
RSLT = POSTF(2,3,497,TULNUM)      $$ Write in the new value
```

The above line would be read as follows:

Increment **TULNUM** by 1, Then set DBLCOM 497 with the current value of the variable **TULNUM**.

5.3 Function Type 03 (Set COMMON to Empty)

***rslt* =POSTF(03,*arg1*,*arg2*)**

rslt: zero returned
arg1: type of COMMON
1= integer
2= real
3= double
arg2: COMMON location number to be set to empty.
INTCOM number range is 1 – 3500
RELCOM number range is 1 – 902
DBLCOM number range is 1 – 1000

Function type 03 sets the COMMON value to EMPTY. This is **-32767** for an INTCOM value and **999999.0** for a DBLCOM or RELCOM value. Refer to the appropriate G-Post User's Guide for a description of COMMON locations and contents.

This is an effective way to tell the machine that a particular mode is not available. When you set the COMMON value to empty, it is the same as entering N/A in the option file.

The following example shows how to tell the machine that metric mode is not available. ***Arg2*** is the G-code for metric mode, INTCOM 1864.

RSLT = POSTF(3,1,1864)

The above line would be read as follows:

Set INTCOM 1864 to empty (-32767).

5.4 Function Type 04 (Test for COMMON Empty)

***rslt* =POSTF(04,*arg1*,*arg2*)**

rslt: **0=** COMMON location is not empty
1= COMMON location is empty
arg1: type of COMMON
1= integer
2= real
3= double
arg2: COMMON location number to be tested.
INTCOM number range is 1 – 3500
RELCOM number range is 1 – 902
DBLCOM number range is 1 – 1000

Testing a value for EMPTY is an effective way of determining whether a particular mode is available. This example shows how to see if incremental mode (G-code 505) is available for this machine. If a zero (0) is returned, the value is EMPTY, which means incremental mode is not available.

Refer to the appropriate G-Post User's Guide for a description of COMMON locations and contents.

RSLT = POSTF(4,1,505)

The above line would be read as follows:

Test INTCOM 505, is it empty, Yes **RSLT=1**, No **RSLT=0**.

5.5 Function Type 05 (Get Number of Words in CL Record)

***rslt* = POSTF(05)**

rslt: The number of words in the current CL record

See Chapter 3 for a complete explanation of how FIL handles the CL record structure.

Use Function Type 05 to find out how many words are in the current CL record. In the following example, we are examining the statement **SPINDL/300,CLW**. The result returned is 5.

```
CIMFIL/ON,SPINDL          $$ Trap SPINDL statement
 RSLT=POSTF(20)           $$ Save as current CL record
 NUMWDS=POSTF(5)          $$ Find the number of words
                          $$ NUMWDS = 5
CIMFIL/OFF
```

1	2	3	4	5
CL Record number	Record Type	SPINDL	300	CLW
n	2000	1031	300.0	60

Figure 5-1. CL Record Structure Word Count

The above line would be read as follows:

Count the number of words in the current CL record and set the variable **NUMWDS** to that value.

5.6 Function Type 06 (Find Word Type in CL Record)

***rslt* = POSTF(06,*arg1*)**

rslt: **0=** type is minor word
1= type is a scalar
2= type is a text

arg1: CL record location to be tested

Function Type 06 tells you what type word is in the location you specify with ***arg1***. In our example of **SPINDL/300,CLW**, we specified location 4. The result, 1, tells us that the word in location 4 (300) is a scalar. See Figure 5-1.

You will usually use location 4 or greater, since you already know that location 1 is the CL record number, location 2 is the CL record type, and location 3 is the major word in the CL record (such as the one that you specified with the CIMFIL/ON, statement).

You'll usually use Function Type 06 with Function Type 07 or Function Type 08.

Example:

```
WDTYPE=POSTF(6,4)          $$ Return type of word in
                           $$  CL rec location 4
```

The above line would be read as follows:

Read the fourth word of the current CL record and set the variable **WDTYPE** to the word type 0, 1, or 2.

5.7 Function Type 07 (Get CL Word Value)

***rslt* =POSTF(07,*arg1*)**

rslt: CL word value returned
arg1: CL word location

After having determined the word type with Function Type 06, you can use Function Type 07 to get the value. Using our example of **SPINDL/300,CLW** shown in Figure 5-1, we know that the word in location 4 is a scalar. This statement returns the value, which is 300 in this case.

```
VALUE=POSTF(7,4)     $$ Return the value of CL
                     $$  record location 4
```

The above line would be read as follows:

Get the value from the fourth word of the current CL record and set the variable **VALUE** to that value.

Note: VALUE=POSTF(7,-1) will allow the user to get the CL record number located by the POSTF(30…) command when it reads ahead. The command VALUE=POSTF(7,1) always returns the current CL record number.

5.8 Function Type 08 (Get CL Record Text)

***rslt* = TEXT/CLW**

rslt: text variable (maximum 66 characters)

If the result returned with Function Type 06 told you that the word type was text, you can use Function Type 08 (**TEXT/CLW**) to read the text. This is a practical way to return the contents of **PARTNO**, **PPRINT**, or **INSERT** statements from the current CL record.

Figure 5-2 shows the CL record structure for the post processor statement **PARTNO/'TEST ∴THIS'** . Each location contains two characters
(a space is considered to be one character) For the sake of illustration, we show a space as∴

1	2	3	4	5	6	7	8
CL Record	**Record Type**	**PARTNO**					
n	**2000**	**1045**	**TE**	**ST**	**$_b$T**	**HI**	**S$_b$**

Figure 5-2. Text CL Record

The following FIL code returns the text contained in the **PARTNO** statement. In this case, the text is **TEST THIS** .

Example:

```
TXT =TEXT/CLW          $$ Returns text in PARTNO stmt
```

5.9 Function Type 09 (Put a Minor Word in CL Record)

***rslt* =POSTF(09,*arg1*,*arg2*)**

rslt: zero returned
arg1: CL word location to be set
arg2: Integer code of minor word to be set.

Function Type 09 enables you to add or to change a minor word in the CL record.

The first example shows you how to change **SPINDL/300,CLW** to **SPINDL/300,CCLW**. You've already used Function Types 06 and 07 to determine that the word in location 5 is a scalar, 60. This code changes that word to 59 (CCLW):

```
RSLT = POSTF(9,5,59)
```

The above line would be read as follows:

Set location or word 5 of the current CL record to the integer code 59, which is the integer code for **CCLW**. Then set **RSLT** to zero (0) to indicate success.

Remember that you can also use **ICODEF** to get the integer code of a word. You could use the following line to perform the same task:

```
RSLT = POSTF(9,5,(ICODEF(CCLW)))
```

Notice that by assigning a new value to the word in location 5, you overwrote the previous value. See Figure 5-3.

1	2	3	4	5
CL Record number	Record Type	SPINDL	300	CLW
n	2000	1031	300.0	60

1	2	3	4	5
CL Record number	Record Type	SPINDL	300	CCLW
n	2000	1031	300.0	59

Figure 5-3. Change Minor Word in CL Record

The next example shows you how to add a minor word to the CL record. We're going to change our faithful **SPINDL/300,CLW** statement to **SPINDL/300,CLW,RANGE,1**. This is a three-step process. The first uses Function Type 09 to add the minor word **RANGE** in location 6. The second step uses Function Type 10 to add the value 1. The final step is to use Function Type 12 to reset the number of words in the CL record.

RSLT = POSTF(9,6,(ICODEF(RANGE))) $$ Add word at location 6

1	2	3	4	5	6
CL Record number	**Record Type**	**SPINDL**	**300**	**CLW**	**RANGE**
n	**2000**	**1031**	**300.0**	**60**	**145**

Figure 5-4. Add Minor Word to CL Record

Figure 5-4 shows how the CL record changes when you add a minor word.
In our example, we placed RANGE in location 6 because we know our CL record contained 5 words (and therefore ended at location 5). It's just like making a train out of building blocks.

Next, we need to add the value 1 to the statement. That's explained on the next page.

5.10 Function Type 10 (Put a Scalar in CL Record)

***rslt* =POSTF(10,*arg1*,*arg2*)**

rslt: zero returned

arg1: CL word location to be set.

Arg2: scalar value to store

This is a continuation of the **SPINDL/300,CLW,RANGE,1** example shown on the previous pages.

We just used Function Type 09 to add the minor word **RANGE** in location 6. Now we're going to add the value (1) in location 7.

```
RSLT = POSTF(10,7,1)          $$ Add scalar to location 7
```

Figure 5 shows what our CL record looks like now.

1	2	3	4	5	6	7
CL Record Number	**Record Type**	**SPINDL**	**300**	**CLW**	**RANGE**	**1**
n	**2000**	**1031**	**300.0**	**60**	**145**	**1.0**

Figure 5-5. Add Scalar Value to CL Record

You can see that we have effectively changed the word count in the CL record from 5 to 7. However, the pointer in the CL file expects to find 5 words in this record. The final step in our example is to use **POSTF** Function Type 12 to reset the number of words in the CL record.

5.11 Function Type 11 (Unused)

Note: This function is reserved for future use.

5.12 Function Type 12 (Set Number Words in CL Record)

***rslt* =POSTF(12,*arg1*)**

rslt: zero returned
arg1: number of words in CL record

Any time you increase (or decrease) the number of minor words and/or scalars in a CL record, you must use Function Type 12 to reset the word count.

When we changed **SPINDL/300,CLW** to **SPINDL/300,CLW,RANGE,1,** we changed the actual word count from 5 to 7. However, there is a pointer in the CL file that is expecting 5 words. Unless we change that pointer, the words we added will be ignored.

Example:

```
RSLT = POSTF(12,7)   $$ Change word count to 7
```

The above line would be read as follows:

Reset the number of words in the current CL record to 7.

5.13 Function Type 13 (Process Current CL Record)

***rslt* =POSTF(13)**

rslt: zero returned

This function sends the current CL record to the post for processing.

There are times when you don't want to send the record to the post. For example, you might have added words to the CL file that the post would not recognize. If you processed that record, the post would give you warning messages.

Example:

```
RSLT = POSTF(13)  $$ PROCESS THE CURRENT  CL RECORD
```

Any time you want to send a CL record to the post processor you must use this function. If you use this function more than once in a FIL subroutine the current CL record will be executed each time the function is encountered.

5.14 Function Type 14 (Read Next CL Record From CL File)

***rslt* = POSTF(14,*arg1*)**

rslt:	**=0**	read successful
	=1	read error
arg1:	**=1**	optional to return any next record. If not given, return the next non-**TYPE 1000** record

You will probably want to use **POSTF(14)** rather than **POSTF(14,1)** most of the time. If you use ***arg1***, you'll get the next record. If you omit ***arg1***, you'll get the next non-**TYPE 1000** record.

Example:

Read next Non-TYPE 1000 CL record.

```
RSLT=POSTF(14)      $$ Return the next non-
                    $$  Type 1000 record
```

Read next CL record.

```
RSLT=POSTF(14,1)    $$ Return the next CL
                    $$  record
```

When using this function to read additional CL records, each time the next CL record is read, it becomes the current CL record and the previous CL record is lost, unless it had been saved with function type 20.

5.15 Function Type 15 (Position to a CL Record in CL File)

***rslt* =POSTF(15,*arg1*) (Original format)**

rslt: **=0** file position successful
=1 file position error
arg1: CL record number to position

This function enables you to position to a specific CL record number.

If this form POSTF(15,n) is operated on a GOTO/cmd with a very large CL file (like 30 thousand points), then the I/O time taken may be very large and your program can run for a long time – possibly hours.

Using the POSTF(15,0,1-2) (Fast I/O) form, you save the file info at a known location and then restore it back when the read ahead is completed. This may reduce your run time significantly – from hours to minutes.

Remember that you can use function type 7 to find the CL record number.

Warning: **Be careful when you use this function. If you enter the wrong number, you could get unexpected results. For example, if you skip some CL record numbers, the post will leave out that part of the file.**

If you position to a record that comes before the record on which you were working, the post will go into an infinite loop. If the post is unattended, you run the risk of filling all available space on your disk.

***Rslt* =POSTF(15,*arg1,arg2*) (Fast I/O format) (Windows Platforms Only)**

rslt: **=0** file position successful
=1 file position error
arg1: **=0** Specifies Fast I/O
arg2: **=1** Saves the current CL file pointer location
arg2: **=2** Re-Positions the CL file pointer to the last saved location

This Fast I/O format greatly increases the processing speed when numerous CL read ahead's are used. If you are working in a FIL routine such as CIMFIL/ON,GOTO that requires a read ahead and you are processing large CL files, using the Fast I/O format could reduce processing time from hours to minutes.

Note: **It is not necessary to convert your existing FIL file read ahead's to this format unless you are experiencing unusually long processing times**

Fast I/O is only available on the Windows Platforms.

Example:

Assume you want examine each GOTO/cmd for duplicate X-values.

Regular POSTF(15,n) format:

This format uses the regular, POSTF(15) and if you observe a **long run time with large CL files**, then try the new **Fast I/O** method .

```
CIMFIL/ON,GOTO
  IR1=POSTF(7,1)              $$ SAVE RECD NUMBER
  DMY=POSTF(13)               $$ EXECUTE GOTO
  X1=POSTF(7,6)               $$ SAVE X-VALUE1
  DMY=POSTF(14)               $$ READ NEXT GOTO
  X2=POSTF(7,6)               $$ SAVE X-VALU2
  IF(X2 .EQ. X1) THEN         $$ COMPARE
    PPRINT/'SAME X-FOUND'
  ENDIF
  DMY=POSTF(15,(IR1+1))       $$ REPOSITION CLFILE
CIMFIL/OFF
```

New Fast IO POSTF(15,0,1-2) format:

```
CIMFIL/ON,GOTO
   DMY=POSTF(15,0,1)          $$ SAVE CLFILE INFO
   DMY=POSTF(13)              $$ EXECUTE GOTO
   X1=POSTF(7,6)              $$ SAVE X-VALUE1
   DMY=POSTF(14)              $$ READ NEXT GOTO
   X2=POSTF(7,6)              $$ SAVE X-VALU2
   IF(X2 .EQ. X1) THEN        $$ COMPARE
     PPRINT/'SAME X-FOUND'
   ENDIF
   DMY=POSTF(15,0,2)          $$ RESTORE CLFILE INFO
CIMFIL/OFF
```

5.16 Function Types 16, 17, 18 (Unused)

Note: These functions are reserved for future use.

5.17 Function Type 19 (Output Current Post Block)

***rslt* = POSTF(19)**

rslt: Output post block

This sends an End-of-Block notation to the post processor and any pending output will be output. All output after this function statement goes into a new block.

Example:

```
RSLT = POSTF(19)      $$ OUTPUT ANY PENDING DATA
```

5.18 Function Type 20 (Save Current CL Record)

***rslt* =POSTF(20)**

rslt: Current CL record is saved for later use.

We strongly suggest that you use this function immediately after each **CIMFIL/ON**, statement.

For example, you position to a record with the **CIMFIL/ON** statement
and begin working with the information in that statement. As soon as you issue a post processor command, such as **COOLNT/ON,** that command becomes the current CL record and the previous CL record is lost. That's fine if that's what you're expecting. However, you can get into some real problems if you think you're working with one record when you're actually working with a different one. This function saves the current CL record in a buffer so you can return to it with function type 21.

In the following example, we identified the first **SPINDL** statement as being the current CL record. We manipulated the data in the CL record, then we issued a Post statement, **POSTN**. However, we had more work we wanted to do with the **SPINDL** CL record. So we used **POSTF(21)** to reload the saved **SPINDL** CL record as the current CL record.

Example:

```
 CIMFIL/ON,SPINDL                  $$ Trap the SPINDL CL Records
  RSLT=POSTF(20)                   $$ Save the current CL record (SPINDL)
                                   $$ in the save buffer
        .
  POSTN/OUT,7,0,24,50,25,50   $$ Send POSTN to the post
                                   $$ POSTN becomes the current CL
                                   $$ record
  RSLT=POSTF(21)                   $$ Reload saved SPINDL record as
                                   $$ current
  RSLT=POSTF(13)                   $$ Execute the current CL record
CIMFIL/OFF
```

5.19 Function Type 21 (Load a Saved CL Record)

***rslt* =POSTF(21)**

rslt: Saved CL record from last function 20 is set as current CL record

This function restores the last CL record saved by function 20.

If this function is used prior to using a function 20 an error will occur.

Example:

```
 CIMFIL/ON,SPINDL                  $$ Trap the SPINDL CL Records
  RSLT=POSTF(20)                   $$ Save the current CL record (SPINDL)
                                   $$ in the save buffer
          .
  POSTN/OUT,7,0,24,50,25,50   $$ Send POSTN to the post
                                   $$ POSTN becomes the current CL
                                   $$ record
  RSLT=POSTF(21)                   $$ Reload saved SPINDL record as
                                   $$ current
  RSLT=POSTF(13)                   $$ Execute the current CL record
CIMFIL/OFF
```

5.20 Function Type 22 (Get Current Machine Number)

***rslt* =POSTF(22)**

rslt: Machine number of the post being executed.

You'll find this function useful if you use multiple post processors from within FIL.

Example:

```
MACHNO = POSTF(22) $$ GET THE CURRENT POST NUMBER
IF(MACHNO .EQ. 10)THEN
  PPRINT/' USING POST #10'
ESLE
  PPRINT/' USING UNKNOWN POST #'
ENDIF
```

5.21 Function Type 23 (Move COMMON Values)

***rslt* = POSTF(23,*arg1*,*arg2*,*arg3*)**

rslt: zero returned
arg1: type of COMMON
1= integer
2= real
3= double
arg2: COMMON location number to load new value into
INTCOM number range is 1 – 3500
RELCOM number range is 1 – 902
DBLCOM number range is 1 – 1000
arg3: COMMON location from which value is moved
INTCOM number range is 1 – 3500
RELCOM number range is 1 – 902
DBLCOM number range is 1 – 1000

This function enables you to move COMMON values from one location to another. Both values must be the same type of variable INTCOM, RELCOM, or DBLCOM.

Refer to the appropriate G-Post User's Guide for a description of COMMON locations and contents.

In this example, we're moving INTCOM 1966 to INTCOM 1986.

Example:

```
RSLT = POSTF(23,1,1986,1966)
```

5.22 Function Type 24 (Trace On/Off)

***rslt* = POSTF(24,*arg1*)**

rslt: zero returned
arg1: trace flag
=**0** turn off trace, print CL ISN numbers
=**1** turn on trace, print FIL file ISN numbers

Use this function to control the trace.

When FIL processes the filter file for syntax, it assigns a negative ISN number to the input lines. This gives you a method of quickly determining whether the input was from the CL file or the FIL file.

This is a handy debug mechanism for all FIL routines. Insert the following debug statements in the global area of your FIL file.

Example:

```
$$ PRINT/OFF,IN                    $$ USE FOR PRODUCTION
$$ DMY = POSTF(24,0)               $$ USE FOR PRODUCTION
DMY = POSTF(24,1)                  $$ USE FOR TESTING
PRINT/ON                           $$ USE FOR TESTING
```

Note:	**When testing and debugging your FIL file comment the first two lines above, as shown. When the post processor is ready to turn over to production comment the third and fourth lines and remove the comments from the first two lines.**

5.23 Function Type 25 (Redirect Post Output)

***rslt* = POSTF(25,*arg1 [,arg2] [,arg3]*)**

rslt: zero returned
arg1: direction flag:
- =**0** normal Post output
- =**1** redirect output to auxiliary file
- =**2** rewind auxiliary file
- =**3** specify a file name for the auxiliary file, ******* See notes on the next page.

Arg2: Optional – File choice flag:
- =**0** Default, redirect output to both the MCD and Listing files
- =**1** redirect output to the Listing file only
- =**2** redirect output to the MCD file only

arg3: Optional – File name for auxiliary file, ******* See notes on the next page.

The post processor normally outputs data to two files, the MCD (.PU1) and the listing file (.LST). When using this function the post processor output is redirected to two auxiliary or temporary files and the output to the punch and listing file is terminated.

Use TEXT/READ,PUNCH or TEXT/READ,PRINT to access the data from the auxiliary files. The following example illustrates this concept.

Example:

```
 CIMFIL/ON,SPINDL              $$ Capture the Spindle CL Records
   I1=POSTF(19)                $$Dump any pending blocks
   I1=POSTF(25,1)              $$Redirect print,punch to auxiliary file
     I1=POSTF(13)              $$Execute current command
     I1=POSTF(19)              $$Force output
   I1=POSTF(25,0)              $$Set Post print,punch to normal
  T1=TEXT/READ,PUNCH           $$Get last punch block
  T2=TEXT/'/ABC/',T1           $$Edit punch line by
                               $$adding '/ABC/'
  SEQNO/OFF                    $$Turn off sequence number
  INSERT/T2                    $$Send in edited line
  SEQNO/ON                     $$Turn on sequence number
CIMFIL/OFF
```

Instructions for using arg1 option 3 "POSTF(25,3,1-2,T1)":

This format allows you to specify their own file name via a text variable in **arg3** instead of the default auxiliary file name. In this form all the four arguments **must** be given, **arg1** must be set to **3** and **arg2**, the print or punch file indicator, must be set to **1** or **2**.

Example:

Let us say you want to send the punch output generated for the CL data between the DEFSUB-ENDSUB CL records to a separate file named **subr01.tap** to define a subroutine on the machine control unit.

```
$$ START RE-DIRECTION OF TAPE DATA TO FILE
CIMFIL/ON,DEFSUB
  T1=TEXT/'subr01.tap'
  DMY=POSTF(25,3,2,T1)
  DMY=POSTF(25,1)
  DMY=POSTF(13)
  DMY=POSTF(25,0)
CIMFIL/OFF

$$ END RE-DIRECTION OF TAPE DATA TO FILE
CIMFIL/ON,ENDSUB
  DMY=POSTF(13)
  DMY=POSTF(25,0)
CIMFIL/OFF
```

With the new **arg1** option **3**, you do not have to read and write the auxiliary punch file using the TEXT/READ,PUNCH and FILEF(1,1,T1) commands. The punch data is automatically written to a file with the name you specify.

Note: Rules for using your own file name:

1. You must specify all the four (4) arguments.
2. You can only provide a user file name before the POSTF(25,1), start re-direction, command.
3. POSTF(25,0) or POSTF(25,2) will use the last known file name, either user file name or the default auxiliary file name.
4. You must use a different file name for a 2nd re-direction. If not, the contents of the first file will be replaced.

Warning:	**Using this function incorrectly can cause the punch and listing files to be empty.**

5.24 Function Type 26 (Control CIMFIL)

***rslt* = POSTF(26, *arg1,arg2,arg3*)**

rslt: zero returned
arg1: CL record type (1-14)
arg2: CL record subtype (0=any subtype)
arg3: control flag:
=**0** disable CIMFIL/ON for this type
=**1** enable CIMFIL/ON for this type

This function type addresses the needs of advanced users who want to use FIL with very large CL files. Function Type 26 works something like an **IF-THEN-ELSE** or **CASE-WHEN** statement to find a specific subtype. You can then disable the **CIMFIL/ON** statement after you have found the information you need.

In this example, we show you how to find a Type 5000 motion record, get the first Z-value so you can generate an H-code, then stop looking at motion records. This is especially helpful if you have several thousand lines of codes to examine.

Example:

```
CIMFIL/ON,5,5                  $$ Catch the Type 5000 CL Records
  I1=POSTF(20)                 $$ Save CL record
  ZV=POSTF(7,8)                $$ Get Z value
  INSERT/'G00Z',ZV,'H1'        $$ Output length comp
  I1=POSTF(21)                 $$ Load saved record
  I1=POSTF(13)                 $$ Process record
  I1=POSTF(26,5,5,0)           $$ Disable CIMFIL for
                               $$  subsequent motion records
CIMFIL/OFF
```

5.25 Function Type 27 (Security ID Number)

***rslt* = POSTF(27)**

rslt: Security ID Number

This function returns the system security ID number. This command will return a unique system ID number. For Surfcam users this will return the SIM number.

Example:

```
CIMFIL/ON,MACHIN
   DMY=POSTF(13)                  $$ Execute the MACHIN statement
   SID=POSTF(27)                  $$ Get the security number
   IF (SID.EQ.137602) THEN
         PPRINT/'SECURITY OK'

ELSE
   PPRINT/'SECURITY FAILED'
   DMY=POSTF(9,14000)             $$ Load FINI CLRECORD
   DMY=POSTF(13)                  $$ Execute

ENDIF

CIMFIL/OFF
```

5.26 Function Type 28 (Locate word/scalar/couplet in the CL Record)

***rslt* = POSTF(28, *arg1,arg2*)**

rslt: Location index or zero if not found
arg1: type of input given in ***arg2***
1 = locate index matching the scalar in ***arg2***
2 = locate index matching the minor word in ***arg2***
3 = locate index matching the minor word couplet in ***arg2***
arg2: scalar or word to locate

This function searches the current CL record, starting at location 4 (the first word to the right of the slash (/)) and returns the index location of ***arg2*** or zero if ***arg2*** is not found. The location index will always be in the range of 4 through the number of CL words or zero.

Note: A ***couplet*** is defined as a Minor Word followed by a scalar, such as RANGE,1 or MAXRPM,1500.

Example:

1. CL Record: SPINDL/300,HIGH,RPM,81,CLW,RANGE,1

   To locate the scalar 81:

   IDX = POSTF(28,1,81)
   IDX will be set to 7

2. CL Record: SPINDL/300,HIGH,RPM,LOCK,CLW,RANGE,1

   To locate the word HIGH:

   IDX = POSTF(28,2,(ICODEF(HIGH)))
   IDX will be set to 5

3. CL Record: SPINDL/300,HIGH,RPM,LOCK,CLW,RANGE,1

   To locate the word LOW:

   IDX = POSTF(28,2,(ICODEF(LOW)))
   IDX will be set to 0 as not found

4. CL Record: SPINDL/300,RPM,CLW,RANGE,4

   To locate the RANGE couplet:

   IDX = POSTF(28,3,(ICODEF(RANGE)))
   IDX will be set to 7

Note: Use VAL1 = POSTF(7,(IDX+1)) to get the couplet value 4.

5.27 Function Type 29 (Remove word/scalar/couplet in the CL Record)

***rslt* = POSTF(29, *arg1*,*arg2*)**

rslt: returns zero, automatically adjust the number of CL words.
Arg1: type of input given in ***arg2***
1 = remove CL word location given in ***arg2***
2 = remove the 1st scalar matching ***arg2***
3 = remove the minor word matching ***arg2***
4 = remove the minor word couplet matching ***arg2*** (2 CL words removed)
arg2: location, scalar, minor word or minor word couplet to remove

This function removes CL words from the current CL record and then adjust the length of the CL record.. The CL word can be a location, scalar, minor word or minor word couplet.

When ***arg1*** = 1, the location index, ***arg2***, must be greater than 3 and less than the number of words in the current CL record.

When ***arg1*** = 2, Zero is a valid value for ***arg2***.

When ***arg1*** = 3, Zero is an invalid value for ***arg2*** and will be ignored.

When ***arg1*** = 4, A ***couplet*** is defined as a Minor Word followed by a scalar, such as RANGE,1 or MAXRPM,1500 and thus 2 words will be removed from the CL record.

Note: A ***couplet*** is defined as a Minor Word followed by a scalar, such as RANGE,1 or MAXRPM,1500.

Caution: Once a word is removed, it is important to note the number of words in the current CL record will be automatically adjusted as if you executed a POSTF(12, ***arg1***).

Example:

2. CL Record: SPINDL/300,HIGH,RPM,LOCK,CLW,RANGE,1

   To remove HIGH: DMY = POSTF(29,1,5)

   The resulting CL Record is: SPINDL/300,RPM,LOCK,CLW,RANGE,1

   To remove LOCK: DMY = POSTF(29,1,6)

   The resulting CL Record is: SPINDL/300,RPM,CLW,RANGE,1

3. CL Record: SPINDL/300,RPM,LOCK,81,CLW,RANGE,1

   To remove the scalar 81: DMY = POSTF(29,2,81)

   The resulting CL Record is: SPINDL/300,RPM,LOCK,CLW,RANGE,1

4. CL Record: SPINDL/300,HIGH,RPM,LOCK,CLW,RANGE,1

   To remove HIGH, as in example 1: DMY = POSTF(29,3,(ICODEF(HIGH)))

   The resulting CL Record is: SPINDL/300,RPM,LOCK,CLW,RANGE,1

   To remove LOCK, as in example 1: DMY = POSTF(29,3,(ICODEF(LOCK)))

   The resulting CL Record is: SPINDL/300,RPM,CLW,RANGE,1

5. CL Record: SPINDL/300,RPM,LOCK,81,CLW,RANGE,1

   To remove the LOCK,81 couplet: DMY = POSTF(29,4,(ICODEF(LOCK)))

   The resulting CL Record is: SPINDL/300,RPM,CLW,RANGE,1

   To remove RANGE,1 couplet: DMY = POSTF(29,4,(ICODEF(RANGE)))

   The resulting CL Record is: SPINDL/300,RPM,CLW

Note: A ***couplet*** is defined as a Minor Word followed by a scalar, such as RANGE,1 or MAXRPM,1500.

5.28 Function Type 30 (Read the Next Specified CL Record From the CL File)

***rslt* = POSTF(30,*arg1*,*arg2*) (Short format)**

rslt:	**=0**	success, the desired CL record was found and the desired CL file positioning was applied.
	=1	failed, the desired CL record not found, no CL file positioning was applied.
arg1:		specifies the Integer Code for the CL record type (i.e.1031 for SPINDL)
arg2:	**=0**	re-position the CL file pointer to the original CL record position.
	=1	leave the CL file pointer at the new CL record position. G-Post will read and process this record.
	=2	leave the CL file pointer at the next CL record, following the new CL record position. G-Post will skip and not and process this record.

Note: for ***arg1*** you can use the integer code,1031 for SPINDL, or the function ICODEF(SPINDL). FIL requires an integer code for any major or minor words within a POSTF command.

***Rslt* = POSTF(30,*arg1*,*arg2*,*arg3*[,*arg4*]) (Long format)**

rslt:	**=0**	success, the desired CL record was found and the desired CL file positioning was applied.
	=**1**	failed, the desired CL record not found, no CL file positioning was applied.
arg1:		specifies the Integer Code for the CL record type (i.e.2000 for post commands).
arg2:		specifies the Integer Code for the CL record sub-type (i.e.1031 for SPINDL).
arg3:	**=0**	re-position the CL file pointer to the original CL record position.
	=1	leave the CL file pointer at the new CL record position. G-Post will read and process this record.
	=2	leave the CL file pointer at the next CL record, following the new CL record position. G-Post will skip and not and process this record.
arg4:	**=n**	Optional, limits the search to a number of records (n), 0 reads to the FINI.

Note: for ***arg2*** you can use the integer code1031 or the function ICODEF(). FIL requires an integer code for any major or minor words within a POSTF command.

POSTF(30) function reads the next CL record, of a given type, from the CL file. The alternative to using **POSTF(30)** is using a combination of **POSTF(14)** and **POSTF(15)** along with a **DO**/loop to read ahead in the CL file, this can get complex. **POSTF(30)** simplifies the process of reading ahead in the CL file.

When using this function to read additional CL records, each time the next CL record is read, it becomes the current CL record and the previous CL record is lost, unless it had been saved with **POSTF(20)**.

Example:

Assume the sample input is given below:

```
PARTNO TEST
MACHIN/UNCX01,1
SPINDL/10
FEDRAT/10
LOADTL/1
COOLNT/ON
RAPID
GOTO/10,10,10
CYCLE/DRILL,1,10,IPM,.1
GOTO/2,2,2
GOTO/1,1,1
CYCLE/OFF
RAPID
GOTO/10,10,10
FINI
```

1. Find the next tool number and output a message when processing the SPINDL command.

```
CIMFIL/ON,SPINDL
   DMY=POSTF(13)                        $$ PROCESS SPINDL RECORD
   WRD=ICODEF(LOADTL)
   FLG=POSTF(30,WRD,0)                  $$ LOAD THE CL WITH LOADTL RECD
   IF(FLG .EQ. 0) THEN                  $$ FOUND LOADTL OK
     TLN=POSTF(7,4)                     $$ GET TOOL NUMBER
     PPRINT/'NEXT TOOL NEEDED =',TLN
  ENDIF
CIMFIL/OFF
```

We left the CL record at the sequential position after SPINDL with third ***arg3***=0 value.

2. If current speed is greater than 1000, terminate the job.

```
CIMFIL/ON,SPINDL
   SPD=POSTF(7,4)                       $$ GET SPEED
   IF(SPD .GT. 1000) THEN
     PPRINT/'YOU ARE GOING TOO FAST!, STOP'
     FLG=POSTF(30,14,0,1)               $$ FIND FINI
  ENDIF
CIMFIL/OFF
```

We want FINI to be the next the CL record and also shown is the long format which is same as **FLG=POSTF(30,(ICODEF(FINI)),1)**.

3. Find the command after LOADTL/cmd and process it.

```
CIMFIL/ON,SPINDL
   DMY=POSTF(13)                          $$ PROCESS SPINDL RECORD
    WRD=ICODEF(LOADTL)
    FLG=POSTF(30,WRD,2)                   $$ LOAD THE CL WITH LOADTL RECD
    IF(FLG .EQ. 0) THEN                   $$ FOUND LOADTL OK
      TLN=POSTF(7,4)                      $$ GET TOOL NUMBER
      PPRINT/'SKIP TO CMD AFTER NEXT TOOL =',TLN
    ENDIF
 CIMFIL/OFF
```

We left the CL record at the sequential position after LOADTL with third ***arg3***=2 value. G-Post will process COOLNT/ON as the next command.

5.29 Function Type 31 (_OUTPT Macro Functions)

The **POSTF(31)** functions are designed to work within the _OUTPT macro and allow you to manipulate the WORD buffer contents and output. Attempting to use these functions outside of the _OUTPT macro will generate a FIL error. Also see the _OUTPT section of this manual for more details.

***Rslt* =POSTF(31,*arg1,arg2,arg3*)**

arg1: type of function
- **1= Get WORD Value**
- **2= Set WORD Value**
- **3= Empty WORD Buffer**
- **19= Process the WORD Buffer (call G-Post "Output" function)**
- **20= Save WORD Buffer**
- **21= Re-Load the saved WORD Buffer**

arg2: Address location 1-26 A-Z or 27-52 Verify A-Z
arg3: Value to be stored in the WORD Buffer when arg1=2

rslt: zero returned or value retrieved when arg1=1

5.29.1 POSTF(31,1,*arg2*) (Get a Value from WORD)

This POSTF function is used to GET (retrieve) a value from the WORD buffer for the desired letter address.

***Rslt* = POSTF(31,1,*arg2*)**

rslt: **=** will contain the current value stored in the WORD buffer for the desired ***arg2*** address. If the desired address is empty then ***RSLT*** will be set to 999999.

Arg2: **=** **1-52** specifies the desired address 1-26 for A-Z and 27-52 for Verify A–Verify Z.

5.29.2 POSTF(31,2,*arg2,arg3*) (Set a Value in WORD)

This POSTF function is used to SET (load) a value in the WORD buffer for the desired letter address.

***Rslt* = POSTF(31,2,*arg2,arg3*)**

rslt: **=** ***RSLT*** will be set to Zero

arg2: **=** **1-52** specifies the desired address 1-26 for A-Z and 27-52 for Verify A–Verify Z.

arg3: **=** **n** specifies the desired value to be loaded into the WORD buffer for the desire ***arg2*** address.

5.29.3 POSTF(31,3) (Clear the WORD Buffer)

This POSTF function is used to clear the entire WORD buffer. All WORD buffer locations will be set to empty (99999.).

rslt **=** **POSTF(31,3)**

rslt: **=** ***RSLT*** will be set to Zero

5.29.4 POSTF(31,19) (Process the Current WORD Buffer)

This POSTF function is used execute the "Output" function of the G-Post. It will process the current WORD buffer.

Rslt **=** **POSTF(31,19)**

rslt: **=** ***RSLT*** will be set to Zero

5.29.5 POSTF(31,20) (Save the Current WORD)

This POSTF function is used save a temporary copy of the WORD buffer.

Rslt **=** **POSTF(31,20)**

rslt: **=** ***RSLT*** will be set to Zero

5.29.6 POSTF(31,21) (Reload the Saved WORD)

This POSTF function is used reload the WORD buffer with the contents of the last saved copy of the WORD buffer from a previous POSTF(31,20) command.

Rslt **=** **POSTF(31,21)**

rslt: **=** ***RSLT*** will be set to Zero

Sample _OUTPT Macro:

FIL file: (see file, /CAMLIB/_OUTPT.FIL, supplied with the system)

```
$$ 06-15-04 VEN SUDHAKAR @ AUSTIN N.C., INC.; CREATE _OUTPT SAMPLE MACRO
$$
$$ G-POST WILL CALL THIS MACRO JUST BEFORE PROCESSING AN OUTPUT
$$ BLOCK WHEN I4667=1. THIS EDIT MACRO IS DIFFERENT FROM _MCDWT
$$ AS YOU ARE WORKING WITH OUTPUT BLOCK SCALARS INSTEAD OF TEXT
$$ AND CAN AVOID STRING PARSING.
$$
$$ USING THIS MACRO YOU CAN:
$$
$$ 1. TO EDIT THE CURRENT BLOCK WITH ANY POSTF()
$$ 2. OUTPUT ADDITIONAL BLOCKS WITH POSTF(31,19) OR FILEF(4,1,TEXT)
$$ 3. SKIP THE CURRENT BLOCK
$$
$$ NOTE:
$$
$$ 1. ONLY A POST OR GOTO COMMAND THAT TRIGGERED OUTPUT
$$    INCLUDING POSTN CAN BE EDITED
$$ 2. STRING OUTPUT LIKE INSERT,PPRINT ARE BY PASSED
$$ 3. POSTF(31) IS ALLOWED ONLY WITHIN _OUTPT MACRO
$$ 4. REPLAC/CMD IS APPLIED AFTER _OUTPT MACRO
$$ 5. YOU CAN USE ANY FIL COMMAND,POSTF,FILEF INSIDE THIS MACRO
$$ 6. ALL POST COMMANDS INSIDE THE MACRO ARE IGNORED INCLUDING
$$    INSERT OR POSTF(13).
$$
$$ THIS SAMPLE MACRO HAS 4-EXAMPES TO ALTER/ADD/SKIP BLOCKS
$$
$$ EXAMPLE-1 FOR TOOLCHG M06 OUTPUT N9999 G28 X0 Y0 Z0 BEFORE
$$ EXAMPLE-2 CHANGE C-270 TO C90
$$ EXAMPLE-3 FOR G43 ADD Z-CURRENT VALUE IF Z IS NOT THERE
$$ EXAMPLE-4 SKIP G93 BLOCK FROM OUTPUT
$$
REDEF/ON
$$
EMT=999999                              $$ DEFINE EMPTY
EPS=0.00001                             $$ SMALL TOL FOR IF TEST
CONTRL/TOLER,IF,EPS                     $$ SET IF/EQ TEST TOLERANCE
_OUTPT=MACRO/
  MWRD=POSTF(31,1,13)                   $$ 1)CHK M06 GET M-WORD
  IF(MWRD .EQ. 6) THEN
    DMY=POSTF(31,20)                    $$ SAVE M06 BLOCK
    DMY=POSTF(31,03)                    $$ CLEAR BLOCK
    DMY=POSTF(31,2,14,9999)             $$ SET N9999
    DMY=POSTF(31,2,7,28)                $$ G28 X0 Y0 Z0
    DMY=POSTF(31,2,24,0)
    DMY=POSTF(31,2,25,0)
    DMY=POSTF(31,2,26,0)
    DMY=POSTF(31,19)                    $$ OUTPUT G28 BLOCK
    DMY=POSTF(31,21)                    $$ RESTORE M06 BLOCK
    JUMPTO/LB80
  ENDIF
```

```
CWRD=POSTF(31,1,3)                  $$ 2)CHK C-270 GET C-WORD
  IF(CWRD .EQ. (-270)) THEN         $$ CHG C-270 TO C90
    DMY=POSTF(31,2,3,90)
  ENDIF
  GWRD=POSTF(31,1,7)                $$ 3)CHK G43 GET G-WORD
  IF(GWRD .EQ. 43) THEN
    ZWRD=POSTF(31,1,26)             $$ GET Z-WORD
    IF(ZWRD .EQ. EMT) THEN          $$ IF NO Z SET CURRENT Z
      ZCUR=POSTF(1,3,(291+26))
      DMY=POSTF(31,2,26,ZCUR)
    ENDIF
    JUMPTO/LB80
  ENDIF
  IF(GWRD .EQ. 93) THEN             $$ 4)SKIP A G93 BLOCK
    JUMPTO/LB90
  ENDIF
  LB80)CONTIN
  DMY=POSTF(31,19)                  $$ OUTPUT CURRENT BLOCK
  LB90)CONTIN                       $$ BRANCH LB90 TO SKIP OUTPUT
TERMAC
```

5.30 Function Type 32 (Store/Retrieve Scalar from Large Memory Arrays)

The **RESERV/cmd** limits the size of FIL arrays. **POSTF(32)** will allow you to store/retrieve scalars from large memory arrays. The number if arguments will depend on the **arg1** type. Unused arguments can be zero. The function will return zero in **rslt**.

***rslt* = POSTF(32,*arg1*,*arg2*,*arg3*,*arg4*)**

rslt: **=0** successful execution or value of array position read.
=1 failed, the desired array was not opened, read, written to or closed.

arg1: Type of memory operation

=1 open/allocate a memory array
- **arg2:** name of array
- **arg3:** size of array
- **arg4:** not used

=2 store a scalar into a memory array
- **arg2:** name of array
- **arg3:** index of array to store
- **arg4:** scalar value to store

=3 retrieve a scalar from memory array
- **arg2:** name of array
- **arg3:** index of array to retrieve
- **arg4:** scalar variable to update

=4 close a memory array
- **arg2:** name of array
- **arg3:** not used
- **arg4:** not used

FIL Example:

```
$$ REGULAR FIL SYNTAX
RESERV/MYDAT,20                    $$ 20,000 IS NOT ALLOWED TO RESERV
MYDAT(4)=41
D41=MYDAT(4)

$$ NEW POSTF(32,…) SYNTAX
MYDAT=0                            $$ REGISTER NAME IN FIL – 1ST TIME
DMY=POSTF(32,1,MYDAT,20000)        $$ DEFINE MYDAT ARRAY 20,000-LONG
DMY=POSTF(32,2,MYDAT,4,41)         $$ SET MYDAT(4)=41
DMY=POSTF(32,3,MYDAT,4,D41)        $$ GET D41=MYDAT(4)
DMY=POSTF(32,2,MYDAT,14500,51)     $$ SET MYDAT(14500)=51
DMY=POSTF(32,3,MYDAT,14500, D51)   $$ GET D51=MYDAT(14500)
$$ IF YOU NO LONGER PLAN TO USE MYDAT THEN CLOSE AND RELEASE USED
$$ MEMORY ON YOUR SYSTEM. CLOSE IS OPTIONAL AND IS NOT NEEDED
DMY=POSTF(32,4,MYDAT)              $$ CLOSE MEMORY
```

> **Note:** Use **POSTF(32)** only if you plan to use a large array size otherwise, the **RESERV/cmd** is much easier to use. When using **arg1=3**, retrieve, make sure you have stored a value first with **arg1=2**. If not you may get fatal memory errors from your operating system and/or cause process termination.

5.31 Function Type 33 (Store/Retrieve Text String from Large Memory Arrays)

The **RESERV/cmd** limits the size of FIL arrays. **POSTF(32)** will allow you to store/retrieve text strings from large memory arrays. The number if arguments will depend on the **arg1** type. Unused arguments can be zero. The function will return zero in **rslt**.

***rslt* = POSTF(33,*arg1*,*arg2*,*arg3*,*arg4*)**

rslt: **=0** successful execution or value of array position read.
=1 failed, the desired array was not opened, read, written to or closed.

arg1: Type of memory operation

=1 open/allocate a memory array
- **arg2:** name of array
- **arg3:** size of array
- **arg4:** not used

=2 store a string of text into a memory array
- **arg2:** name of array
- **arg3:** index of array to store
- **arg4:** text variable to store

=3 retrieve a string of text from memory array
- **arg2:** name of array
- **arg3:** index of array to store
- **arg4:** text variable to update

=4 close a memory array
- **arg2:** name of array
- **arg3:** not used
- **arg4:** not used

FIL Example:

```
$$ REGULAR FIL SYNTAX
RESERV/MYTXT,20                    $$ 20,000 IS NOT ALLOWED TO RESERV
MYTXT(4)=TEXT/'HI'
T41=TEXT/MYTXT(4)

$$ NEW POSTF(33,…) SYNTAX
MYTXT=TEXT/'0'                     $$ REGISTER NAME IN FIL – 1ST TIME
DMY=POSTF(33,1,MYTXT,20000)        $$ DEFINE MYTXT ARRAY 20,000-LONG
DMY=POSTF(33,2,MYTXT,4,(TEXT/'HI'))        $$ SET MYTXT(4)=TEXT/'HI'
DMY=POSTF(33,3,MYTXT,4,T41)        $$ GET T41=TEXT/MYTXT(4)
T51=TEXT/'ANY TEXT'
DMY=POSTF(33,2,MYTXT,14500,T51)    $$ SET MYTXT(14500)=TEXT/'ANY TEXT'
DMY=POSTF(33,3,MYTXT,14500,T51)    $$ GET T51=TEXT/MYTXT(14500)
$$ IF YOU NO LONGER PLAN TO USE MYTXT THEN CLOSE AND RELEASE USED
$$ MEMORY ON YOUR SYSTEM. CLOSE IS OPTIONAL AND IS NOT NEEDED
DMY=POSTF(33,4,MYTXT)              $$ CLOSE MEMORY
```

Note: Use **POSTF(33)** only if you plan to use a large array size otherwise, the **RESERV/cmd** is much easier to use. When using **arg1=3**, retrieve, make sure you have stored a value first with **arg1=2**. If not you may get fatal memory errors from your operating system and/or cause process termination.

5.32 Function Type 34 (Let G-Post Sleep for n-Seconds)

***rslt* = POSTF(34,*arg1*)**

rslt: **=0** successful execution
=1 failed
arg1: = Number of seconds to sleep.

This function may be useful when used with **SPWNF(…)** function to wait for an external process to finish. When the spawned function is running, you may want to put the G-Post to sleep for a few seconds and then check for the completion status of the spawned process.

One of the completion status methods is via a green flag file like the one used in the old single track rail roads. That is, G-Post/FIL will make a temp file in the current directory (i.e. **grnflg.dat**) which will be deleted by the spawned process at completion. During the processing, you can wait using **POSTF(34,…)**.

FIL Example:

```
CIMFIL/ON,LOADTL
...
T1=TEXT/'GRNFLG.DAT'          $$ DEFINE THE TEMP FILE NAME
I1=FILEF(1,7,T1)              $$ CHECK FLAG FILE
IF(I1 .EQ. 1) THEN            $$ CHECK IF FILE EXISTS
    DMY=FILEF(1,2,T1)         $$ OPEN AND DELETE IF FILE EXISTS
    DMY=FILEF(1,6,T1)
ENDIF
DMY=FILEF(1,3,T1)             $$ OPEN AS NEW FILE
DMY=FILEF(1,1,T1)             $$ WRITE A LINE
DMY=FILEF(1,5,T1)             $$ CLOSE FILE
T1=TEXT/'C:\MYDATABASE\DBTUL.EXE'      $$ TOOL UTILITY
DMY=SPAWNF(T1,2)              $$ RUN DBTUL.EXE NO-WAIT
LB30)CONTIN
DMY=POSTF(34,5)               $$ SLEEP 5-SECS
I1=FILEF(1,7,T1)              $$ CHECK FLAG FILE
IF(I1 .EQ. 1) THEN            $$ CHECK IF FILE EXISTS
    DMY=FILEF(0,1,(TEXT/'DBTUL IS RUNNING')      $$ OUTPUT MESSAGE TO SCREEN
    JUMPTO/LB30
ENDIF
DMY=FILEF(0,1,(TEXT/'DBTUL IS DONE')      $$ OUTPUT MESSAGE TO SCREEN
...
CIMFIL/OFF
```

Function	Type	Syntax	RSLT	*Arguments*
Get CL Info	05	POSTF(05) (Get # of CL Words)	Word Count	
	06	POSTF(06,a1) (Get CL Word Type)	0= mnr wrd 1= scalar 2= text	*a1 = location*
	07	POSTF(07,a1) (Get CL Word Value)	Value	*a1 = location*
	08	TEXT/CLW (Get CL TEXT)	66 char text string	
	28	POSTF(28,a1,a2) (Locate in the CL Record)	Location or 0	*a1 = Type of a2* *1 = scalar* *2 = Minor Word* *3 = Minor Word Couplet* *a2 = scalar or word*
Load CL Info	09	POSTF(9,a1,a2) (Load a minor Word)	0	*a1 = location* *a2 = word*
	10	POSTF(10,a1,a2) (Load a scalar)	0	*a1 = location* *a2 = scalar*
	12	POSTF(12,a1) (Set # of CL Words)	0	*a1 = count of words*
	13	POSTF(13) (Execute CL Record)	0	
	20	POSTF(20) (Save CL Record)	0	
	21	POSTF(21) (Load Saved CL Record)	0	
	29	POSTF(29,a1,a2) (Remove from the CL Record)	0, and adjust CL record length	*a1 = Type of a2* *1 = CL word location* *2 = scalar* *3 = Minor Word* *4 = Minor Word Couplet* *a2 = location, scalar or word*

Table 5-3. Function Type Chart

Function	Type	Syntax	RSLT	Arguments
Post Common	01	POSTF(01,*a1*,*a2)* *(Get Common)*	Value	*a1 = type (1=int,2=rel,3=dbl)* *a2 = location*
	02	POSTF(02,*a1*,*a2*,*a3*) *(Set Common)*	0	*a1 = type (1=int,2=rel,3=dbl)* *a2 = location* *a3 = value*
	03	POSTF(03,*a1*,*a2*) *(Empty Common)*	0	*a1 = type (1=int,2=rel,3=dbl)* *a2 = location*
	04	POSTF(04,*a1*,*a2*) *(Test for Empty)*	0=Unused 1=empty	*a1 = type (1=int,2=rel,3=dbl)* *a2 = location*
	19	POSTF(19) (*Output Post Block)*		
	22	POSTF(22) *(Get Machine Number)*	Machine Number	
	23	POSTF(23,*a1*,*a2*,*a3*) *(Move Common Values)*	0	*a1 = type (1=int,2=rel,3=dbl)* *a2 = New Location* *a3 = Existing Location*
	25	POSTF(25,*a1 [,a2])* *(Redirect Post Output)*	0	*a1 = Direction Flag* *0=Normal* *1=Aux. Files* *2=Rewind Aux. Files* *a2 = [Optional] File Specifier* *0 = Both file* *1 = Listing file only* *2 = MCD file only*
CL File	14	POSTF(14,a1) (Read Next CL Record)	0=OK 1=Error	*a1 = Return next CL record*
	15	POSTF(15,a1) (Position to CL Record) POSTF(15,a1,a2) (Fast I/O – Position to CL Record)	0=OK 1=Error 0=OK 1=Error	*a1 = CL record number* *a1 = 0 (Specifies Fast I/O)* *a2 = 1 Save the current CL file position* *a2 = 2 Rewind CL file to saved position*
	30	POSTF(30,a1,a2) POSTF(30,a1,a2,a3)	0=OK 1=Error 0=OK 1=Error	*a1 = CL record sub-type* *a2 = 0 Reposition to original* *a2 = 1 New position-Process* *a2 = 2 Retain position-Don't Process* *a1 = CL record type* *a2 = CL record sub-type* *a3 = 0 Reposition to original* *a3 = 1 New position-Process* *a3 = 2 Retain position-Don't Process*
	26	POSTF(26,a1,a2,a3) (Control CIMFIL/ON)	0	*a1 = CL Rec Type* *a2 = CL Rec Subtype* *a3 = 0 Disabled* *a3 = 1 Enabled*

Table 5-4. Function Type Chart

Function	Type	Syntax	RSLT	Arguments
Debug	24	POSTF(24,a1) (Trace on/off)	0	*a1 0 Trace off* *a1 = 1 Trace on*
	27	POSTF(27) (Get System ID)	Security ID Number	
	34	POSTF(34,a1) (Put G-Post to sleep)	0	*a1 Number of seconds*
_OUTPT	31	POSTF(31,1,a2) *(Get a WORD Value)*	= Value	*a2 = 1-52 a-z Verify a-z*
		POSTF(31,2,a2,a3) *(Set a WORD Value)*	0=OK 1=Error	*a2 = 1-52 a-z Verify a-z* *a3 = Value to set*
		POSTF(31,3) *(Empty the WORD Buffer)*	0=OK 1=Error	
		POSTF(31,19) *(Output the WORD Buffer)*	0=OK 1=Error	
		POSTF(31,20) *(Save the WORD Buffer)*	0=OK 1=Error	
		POSTF(31,21) *(Reload the WORD Buffer)*	0=OK 1=Error	
Store/ Retrieve data	32	POSTF(32,1,a2,a3,a4) (Store/Retrieve scalars)	0=OK 1=Error	*a2 = Name of array* *a3 = Size of array* *a4 = Not used*
		POSTF(32,2,a2,a3,a4) (Store/Retrieve scalars)	0=OK 1=Error	*a2 = Name of array* *a3 = Index of array to store* *a4 = Scalar variable to store*
		POSTF(32,3,a2,a3,a4) (Store/Retrieve scalars)	0=OK 1=Error	*a2 = Name of array* *a3 = Index of array to retrieve* *a4 = Scalar variable to update*
		POSTF(32,4,a2,a3,a4) (Store/Retrieve scalars)	0=OK 1=Error	*a2 = Name of array* *a3 = Not used* *a4 = Not used*
	33	POSTF(33,1,a2,a3,a4) (Store/Retrieve text)	0=OK 1=Error	*a2 = Name of array* *a3 = Size of array* *a4 = Not used*
		POSTF(33,2,a2,a3,a4) (Store/Retrieve text)	0=OK 1=Error	*a2 = Name of array* *a3 = Index of array to store* *a4 = Text variable to store*
		POSTF(33,3,a2,a3,a4) (Store/Retrieve text)	0=OK 1=Error	*a2 = Name of array* *a3 = Index of array to retrieve* *a4 = Text variable to update*
		POSTF(33,4,a2,a3,a4) (Store/Retrieve text)	0=OK 1=Error	*a2 = Name of array* *a3 = Not used* *a4 = Not used*

Table 5-5. Function Type Chart

6 FIL Examples

Introduction

Here are some interesting examples of ways you can use FIL to modify your post processor output.

Example	Description:
1	A FIL template file.
2	How to throw away a command.
3	How to replace an existing command with another existing command.
4	How to add output to an existing command.
5	How to add a new command.
6	How to enhance an existing command.
7	How to output data at the beginning of the MCD file.
8	How to output data at the end of the MCD file.
9	How to write to an ASCII file.
10	How to read ahead in the CL file.
11	How to output with the first motion after a command.
12	How to change post setting based on other commands.
13	How to read the PARTNO and retrieve information.
14	How to catch the CLEARP commands.
15	How to examine the CL record.
16	How to introduce a new Minor Word in the CL record
17	How to combine codes in an output block
18	How to customize the COOLNT commands
19	How to swap a Minor Word and a Value.
20	The MAD (make address) macros for changing letter address aliases.
21	Remove the Punch File data when an Error occurs.
22	How to support DIMS-CMM data from a PTC NCL file.

6.1 FIL Example 1: Template FIL File:

When you create a new post processor using the Option File Generator a FIL template is used to initialize the FIL file associated with the option file. These FIL templates are stored in the UNC$SYSTEM directory and are named as follows:

Post Processor Type:	FIL Template File Name:
MILL	UNCX01.F00
LATHE	UNCL01.F00
LASER/CONTOURING	UNCC01.F00
WIRE EDM	UNCW01.F00
PUNCH-PRESS	UNCP01.F00

The template FIL files can be modified with the default data you desire. You can setup the format of the template FIL file so each time you create a new post processor the FIL file will be setup to your liking.

Example (Mill Template):

```
$$ ********************************************************
$$ *              Default Mill FIL file created by the OFG              *
$$ ********************************************************

$$  PRINT/OFF,IN              $$ USE FOR PRODUCTION
  PRINT/ON                   $$ USE FOR DEVELOPMENT
  DMY = POSTF(24,1)          $$ USE FOR DEVELOPMENT

$$ ********************************************************
$$ *          G L O B A L   V A R I A B L E   S E C T I O N          *
$$ ********************************************************

  REDEF/ON                        $$ ALLOW VARAIBLES TO BE REDEFINED
$$ THE FOLLOWING VARIABLES ARE DEFINED FOR USE WITH THE
$$ POSTN/ AND REPEAT/ COMMANDS.  THE USE SINGLE LETTER
$$ VARIABLES ANY WHERE ELSE IN FIL WILL REDEFINE THEM
  A=1  ; B=2  ; C=3  ; D=4  ; E=5  ; F=6  ; G=7  ; H=8
  I=9  ; J=10 ; K=11 ; L=12 ; M=13 ; N=14 ; O=15 ; P=16
  Q=17 ; R=18 ; S=19 ; T=20 ; U=21 ; V=22 ; W=23 ;X=24
  Y=25 ; Z=26

$$ ********************************************************
$$ *                  M A C R O   S E C T I O N                  *
$$ ********************************************************

$$ ********************************************************
$$ *                C I M F I L    S E C T I O N                *
$$ ********************************************************

$$ ********************************************************
$$ *                  E N D   O F   F I L   F I L E                *
$$ ********************************************************

FINI
```

6.2 FIL Example 2: How to throw away a command.

```
$$ ********************************************************
$$ *            SAMPLE 1 FIL FILE FOR TRAINING             *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *    A B O U T   T H I S   F I L   F I L E     *
$$ ********************************************************

$$  THIS IS A SAMPLE FIL FILE TO SHOW HOW TO THROW AWAY A COMMAND

$$ ********************************************************
$$ *       G L O B A L   V A R I A B L E   S E C T I O N      *
$$ ********************************************************

 LTSW = 0                                   $$ DEFAULT LOADTL SWITCH
 CNT = 0                                    $$ DEFAULT COUNTER

$$ ********************************************************
$$ *          G L O B A L   M A C R O   S E C T I O N        *
$$ ********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS FROM THE POST
  $$ AND INCREMENTS IT BY ONE
  $$ AND RESETS THE POSTS WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                      $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                               $$ ADD 1
  DMY = POSTF(2,1,1932,TW)                  $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ ********************************************************
$$ *              C O O L N T   S E C T I O N             *
$$ ********************************************************

CIMFIL/ON,COOLNT                            $$ CATCH THE COOLANT COMMANDS
  $$ THROW THE COOLNT COMMANDS AWAY
CIMFIL/OFF

$$ ********************************************************
$$ *              E N D   O F   F I L   F I L E           *
$$ ********************************************************
FINI
```

6.3 FIL Example 3: How to replace an existing command with another existing command.

```
$$ ********************************************************
$$ *            SAMPLE 2 FIL FILE FOR TRAINING             *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *          A B O U T  T H I S  F I L  F I L E          *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO TAKE AN EXISTING COMMAND
$$ GOHOME AND CHANGE ITS OUTPUT USING POSTN OR INSERT

$$ ********************************************************
$$ *      G L O B A L  V A R I A B L E  S E C T I O N     *
$$ ********************************************************

 LTSW = 0                                 $$ DEFAULT LOADTL SWITCH
 CNT = 0                                  $$ DEFAULT COUNTER

$$ ********************************************************
$$ *          G L O B A L  M A C R O  S E C T I O N       *
$$ ********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS
  $$ FROM THE POST AND INCREMENTS
  $$  IT BY ONE AND RESETS THE POSTS
  $$ WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                    $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                             $$ ADD 1
  DMY = POSTF(2,1,1932,TW)                $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ ********************************************************
$$ *            G O H O M E   S E C T I O N              *
$$ ********************************************************

CIMFIL/ON,GOHOME                          $$ CATCH THE GOHOME COMMANDS
  DMY = POSTF(2,1,1867,1)                 $$ POST SIMULATION MODE ON
  DMY = POSTF(13)                         $$ EXECUTE THE CURRENT CL RECORD
  DMY = POSTF(2,1,1867,0)                 $$ POST SIMULATION MODE OFF
  POSTN/OUT,7,28,24,0,25,0,26,0           $$ OUTPUT G28X0Y0Z0
  $$ INSERT/'G28X0.Y0.Z0.$'               $$ COULD USE THIS LINE INSTEAD
CIMFIL/OFF

$$ ********************************************************
$$ *              E N D  O F  F I L  F I L E              *
$$ ********************************************************
FINI
```

6.4 FIL Example 4: How to add output to an existing command.

```
$$ ********************************************************
$$ *            SAMPLE 3 FIL FILE FOR TRAINING            *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *          A B O U T  T H I S  F I L  F I L E          *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO TAKE AN EXISTING COMMAND
$$ LOADTL AND ADD ADDITIONAL OUTPUT LIKE OPSTOP WITH IT

$$ ********************************************************
$$ *       G L O B A L  V A R I A B L E  S E C T I O N    *
$$ ********************************************************

 LTSW = 0                                 $$ DEFAULT LOADTL SWITCH
 CNT = 0                                  $$ DEFAULT COUNTER

$$ ********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N         *
$$ ********************************************************

 TWARN = MACRO/
   $$ THIS MACRO GETS THE TOTAL
   $$ NUMBER OF POST WARNINGS
   $$ FROM THE POST AND INCREMENTS
   $$  IT BY ONE AND RESETS THE POSTS
   $$ WITH THE NEW VALUE.
   TW = POSTF(1,1,1932)                   $$ GET THE TOTAL # OF POST WARNINGS
   TW = TW + 1                            $$ ADD 1
   DMY = POSTF(2,1,1932,TW)               $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ ********************************************************
$$ *              L O A D T L   S E C T I O N             *
$$ ********************************************************

CIMFIL/ON,LOADTL                          $$ CATCH THE LOADTL COMMANDS
  DMY = POSTF(20)                         $$ SAVE THE CURRENT CL RECORD
  OPSTOP                                  $$ OUTPUT OPSTOP M01
  DMY = POSTF(21)                         $$ RELOAD SAVED CL RECORD TO CURRENT
  DMY = POSTF(13)                         $$ EXECUTE THE CURRENT CL RECORD
CIMFIL/OFF

$$ ********************************************************
$$ *                E N D  O F  F I L  F I L E            *
$$ ********************************************************
FINI
```

6.5 FIL Example 5: How to add a new command.

```
$$ ********************************************************
$$ *            SAMPLE 4 FIL FILE FOR TRAINING            *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *          A B O U T  T H I S  F I L  F I L E          *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO ADD A NEW COMMAND
$$ XHOME/value. IF NO VALUE IS SPECIFIED DEFAULT = 30

$$ ********************************************************
$$ *       G L O B A L  V A R I A B L E  S E C T I O N    *
$$ ********************************************************

PPWORD/XHOME,1351                       $$ ADD THE WORD XHOME TO THE FIL SYSTEM,

$$ ********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N         *
$$ ********************************************************

 TWARN = MACRO/
 TW = POSTF(1,1,1932)                   $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                           $$ ADD 1
  DMY = POSTF(2,1,1932,TW)              $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ ********************************************************
$$ *              X H O M E    S E C T I O N             *
$$ ********************************************************

CIMFIL/ON,XHOME                                 $$ CATCH THE XHOME COMMANDS
  XH = 30                                 $$ DEFAULT X HOME LOCATION
  NW = POSTF(5)                           $$ GET THE NUMBER OF CL WORDS
  IF(NW .GT. 3)THEN                       $$ IF VALUE AFTER THE SLASH
   TYP = POSTF(6,4)                       $$ IS IT A MINOR WORD OR VALUE
   IF(TYP.EQ.0)THEN                       $$ ERROR ITS A MINOR WORD
    CALL/TWARN                            $$ RESET THE # OF WARNINGS
    PPRINT/'***WARNING, INVALID XHOME COMMAND'
    JUMPTO/DONE                           $$ GET OUT
   ELSE
    XH = POSTF(7,4)                       $$ GET THE VALUE ENTERED
   ENDIF
  ENDIF
  DMY = POSTF(2,1,1867,1)                 $$ POST SIMULATION MODE ON
  GOTO/XH,0,0                             $$ SET THE POST COMMOND WITH THE COORDINATES
  DMY = POSTF(2,1,1867,0)                 $$ POST SIMULATION MODE ON
  POSTN/ALL                               $$ CLEAR MODAL REGISTERS
  POSTN/OUT,7,0,24,XH,25,0,26,0           $$ OUTPUT G00XvalueY0Z0
  DONE) CONTIN
CIMFIL/OFF

$$ ********************************************************
$$ *              E N D  O F  F I L  F I L E              *
$$ ********************************************************

FINI
```

6.6 FIL Example 6: How to enhance an existing command.

```
$$ ********************************************************
$$ *            SAMPLE 5 FIL FILE FOR TRAINING             *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *           A B O U T  T H I S  F I L  F I L E        *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO ENHANCE A COMMAND
$$ COOLNT. ADD THE FOLLOWING SYNTAX AND OUTPUT
$$
$$ COOLNT/ON              $$ OUTPUT LAST M CODE
$$ COOLNT/OFF             $$ OUTPUT M09
$$ COOLNT/FLOOD           $$ OUTPUT M08
$$ COOLNT/MIST            $$ OUTPUT M07
$$ COOLNT/THRU,LOW        $$ OUTPUT M17
$$ COOLNT/THRU,HIGH       $$ OUTPUT M18
$$ COOLNT/WASH            $$ OUTPUT M53
$$
$$ COOLNT/ON-OFF-FLOOD-MIST ARE EXISTING SYNTAX THE REST WE WILL ADD


$$ ********************************************************
$$ *        G L O B A L  V A R I A B L E  S E C T I O N   *
$$ ********************************************************

 LTSW = 0                                 $$ DEFAULT LOADTL SWITCH
 CNT = 0                                  $$ DEFAULT COUNTER

 PPWORD/WASH,1352                         $$ ADD THE WORD WASH TO THE FIL SYSTEM,
                                          $$ ALSO TO THE APT FILE. 1352 IS THE
                                          $$ INTEGER CODE

$$ ********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N       *
$$ ********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS
  $$ FROM THE POST AND INCREMENTS
  $$  IT BY ONE AND RESETS THE POSTS
  $$  WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                    $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                             $$ ADD 1
  DMY = POSTF(2,1,1932,TW)                $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC
```

```
$$ ******************************************************
$$ *              C O O L N T   S E C T I O N           *
$$ ******************************************************

CIMFIL/ON,COOLNT                          $$ CATCH THE COOLNT COMMANDS
 TYP = POSTF(6,4)                         $$ GET THE CL WORD TYPE 1=VALUE 0=INTEGER CODE
 IF(TYP .EQ. 1)THEN
  DMY = POSTF(13)                         $$ EXECUTE THE CURRENT CL RECORD, THIS IS AN ERROR
 ELSE
  KULTYP = POSTF(7,4)                     $$ GET THE 4TH CL WORD
  CASE / KULTYP                           $$ WHAT IS THE 4TH CL WORD
   WHEN / (ICODEF(ON)),(ICODEF(OFF)),(ICODEF(FLOOD)),(ICODEF(MIST))
    DMY = POSTF(13)                       $$ EXECUTE THE CURRENT CL RECORD
   WHEN / (ICODEF(WASH))                  $$ GOT COOLNT/WASH
    DMY = POSTF(2,1,1962,53)              $$ SET COOLNT/ON TO M53 OUTPUT
    COOLNT/ON                             $$ EXECUTE COOLNT/ON, OUTPUT M53
   WHEN / (ICODEF(THRU))
    NW = POSTF(5)                         $$ GET THE NUMBER OF CL WORDS
    IF(NW.GT.4)THEN
     5TY = POSTF(6,5)                     $$ GET THE CL WORD TYPE,1=VALUE 0=INTEGER CODE
     IF(5TY .EQ. 1)THEN
      DMY = POSTF(13)                     $$ EXECUTE THE CURRENT CL RECORD, ERROR
     ELSE
      5WD = POSTF(7,5)                    $$ GET THE 5TH CL WORD COOLNT/THRU,5TH
      CASE / 5WD                          $$ WHAT IS THE 5TH CL WORD
            WHEN / (ICODEF(LOW))          $$ GOT COOLNT/THRU,LOW
       DMY = POSTF(2,1,1962,17)           $$ SET COOLNT/ON TO M17 OUTPUT
       COOLNT/ON                          $$ EXECUTE COOLNT/ON, OUTPUT M17
            WHEN / (ICODEF(HIGH))         $$ GOT COOLNT/THRU,HIGH
       DMY = POSTF(2,1,1962,18)           $$ SET COOLNT/ON TO M18 OUTPUT
       COOLNT/ON                          $$ EXECUTE COOLNT/ON, OUTPUT M18
            WHEN / OTHERS                 $$ GOT COOLNT/THRU,??? ERROR COMMAND
       DMY = POSTF(13)                    $$ EXECUTE THE CURRENT CL RECORD, ERROR
      ENDCAS
     ENDIF
    ELSE
     DMY = POSTF(13)                      $$ EXECUTE THE CURRENT CL RECORD, ERROR
    ENDIF
   WHEN / OTHERS                          $$ INVALID COOLNT COMMANDS
    DMY = POSTF(13)                       $$ EXECUTE THE CURRENT CL RECORD, ERROR
   ENDCAS
  ENDIF
CIMFIL/OFF

$$ ******************************************************
$$ *              E N D   O F   F I L   F I L E         *
$$ ******************************************************

FINI
```

6.7 FIL Example 7: How to output data at the beginning of the MCD file.

```
$$ *********************************************************
$$ *            SAMPLE 6 FIL FILE FOR TRAINING             *
$$ *********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ *********************************************************
$$ *           A B O U T  T H I S  F I L  F I L E          *
$$ *********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO OUTPUT DATA AT THE BEGINING
$$ OF THE PUNCH FILE. SUCH AS G70G90G40G17 FOLLOWED BY THE PROCESS TIME AND DATE

$$ *********************************************************
$$ *       G L O B A L  V A R I A B L E  S E C T I O N     *
$$ *********************************************************

 LTSW = 0                                 $$ DEFAULT LOADTL SWITCH
 CNT = 0                                  $$ DEFAULT COUNTER

$$ *********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N          *
$$ *********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS
  $$ FROM THE POST AND INCREMENTS
  $$  IT BY ONE AND RESETS THE POSTS
  $$ WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                    $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                             $$ ADD 1
  DMY = POSTF(2,1,1932,TW)                $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ *********************************************************
$$ *              M A C H I N   S E C T I O N              *
$$ *********************************************************

CIMFIL/ON,MACHIN                          $$ CATCH THE MACHIN COMMANDS
 DMY = POSTF(13)                          $$ EXECUTE THE CURRENT CL RECORD
 DMY = POSTF(2,1,1925,3)                  $$ TURN OFF DEFAULT G CODE OUTPUT @ BEGINING
 DMY = POSTF(26,2,(ICODEF(MACHIN)),0)     $$ TURN OFF THIS FIL ROUTINE
 PREFUN/70,NEXT                           $$ OUTPUT G70 WITH NEXT BLOCK
 PREFUN/40,NEXT                           $$ OUTPUT G40 WITH NEXT BLOCK
 PREFUN/90,NEXT                           $$ OUTPUT G90 WITH NEXT BLOCK
 PREFUN/17                                $$ OUTPUT G17G90G70G40
 TIME = TEXT/TIMES                        $$ GET THE CURRENT CPU TIME AND DATE
 PPRINT/'JOB PROCESSED @ ',TIME           $$ OUTPUT TIME AND DATE @ BEGINNING
CIMFIL/OFF

$$ *********************************************************
$$ *                E N D  O F  F I L  F I L E             *
$$ *********************************************************

FINI
```

6.8 FIL Example 8: How to output data at the end of the MCD file.

```
$$ ********************************************************
$$ *            SAMPLE 7 FIL FILE FOR TRAINING            *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *          A B O U T  T H I S  F I L  F I L E          *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO OUTPUT DATA AT THE END
$$ OF THE PUNCH FILE. SUCH AS M02 M30
$$ ALSO THROW AWAY ANY END OR REWIND IN THE CL FILE

$$ ********************************************************
$$ *       G L O B A L  V A R I A B L E  S E C T I O N    *
$$ ********************************************************

 LTSW = 0                                 $$ DEFAULT LOADTL SWITCH
 CNT = 0                                  $$ DEFAULT COUNTER

$$ ********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N         *
$$ ********************************************************

$$ ********************************************************
$$ *              F I N I  S E C T I O N                  *
$$ ********************************************************

CIMFIL/ON,FINI                           $$ CATCH THE FINI COMMANDS (TYPE 14000 CL)
 DMY = POSTF(20)                         $$ SAVE THE CURRENT CL RECORD
 END                                     $$ OUTPUT END OF TAPE M02
 REWIND                                  $$ OUTPUT REWIND M30
 DMY = POSTF(21)                         $$ RELOAD THE SAVED CL RECORD
 DMY = POSTF(13)                         $$ EXECUTE THE CURRENT CL RECORD
CIMFIL/OFF

$$ ********************************************************
$$ *              R E W I N D  S E C T I O N              *
$$ ********************************************************

CIMFIL/ON,REWIND                         $$ CATCH THE REWIND COMMANDS
 $$ THROW AWAY THE REWIND COMMAND
CIMFIL/OFF

$$ ********************************************************
$$ *                E N D  S E C T I O N                  *
$$ ********************************************************

CIMFIL/ON,END                            $$ CATCH THE END COMMANDS
 $$ THROW AWAY THE END COMMAND
CIMFIL/OFF

$$ ********************************************************
$$ *              E N D  O F  F I L  F I L E              *
$$ ********************************************************

FINI
```

6.9 FIL Example 9: How to write to an ASCII text file.

```
$$ ********************************************************
$$ *            SAMPLE 8 FIL FILE FOR TRAINING             *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *          A B O U T   T H I S   F I L   F I L E       *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO WRITE ALL OPERATOR
$$ MESSAGES TO A FILE. THREE FIL SECTION WILL BE USED AS FOLLOWS
$$ MACHIN SECTION TO OPEN THE FILE
$$ PPRINT SECTION TO WRITE TO THE FILE
$$ FINI SECTION TO CLOSE THE FILE

$$ ********************************************************
$$ *       G L O B A L   V A R I A B L E   S E C T I O N  *
$$ ********************************************************

 REDEF/ON                                 $$ ALLOW SYMBOL REDEFINITION
 LTSW = 0                                 $$ DEFAULT LOADTL SWITCH
 CNT = 0                                  $$ DEFAULT COUNTER

$$ ********************************************************
$$ *        G L O B A L   M A C R O   S E C T I O N       *
$$ ********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS
  $$ FROM THE POST AND INCREMENTS
  $$  IT BY ONE AND RESETS THE POSTS
  $$ WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                    $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                             $$ ADD 1
  DMY = POSTF(2,1,1932,TW)                $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ ********************************************************
$$ *                F I N I   S E C T I O N               *
$$ ********************************************************

CIMFIL/ON,FINI                                $$ CATCH THE FINI COMMANDS (TYPE 14000 CL)
 DMY = POSTF(20)                              $$ SAVE THE CURRENT CL RECORD
 IF(FOPEN.EQ.0)THEN                           $$ FILE IS OPEN
  T1 = TEXT/' ************************************************************'
  DMY = FILEF(1,1,T1)                         $$ WRITE STRING TO FILE
  T1 = TEXT/' *     E N D   O F   P P R I N T   F I L E     *'
  DMY = FILEF(1,1,T1)                         $$ WRITE STRING TO FILE
  T1 = TEXT/' ************************************************************'
  DMY = FILEF(1,1,T1)                         $$ WRITE STRING TO FILE
  DMY = FILEF(1,5)                            $$ CLOSE THE FILE
 ENDIF
 DMY = POSTF(21)                              $$ RELOAD THE SAVED CL RECORD
 DMY = POSTF(13)                              $$ EXECUTE THE CURRENT CL RECORD
CIMFIL/OFF
```

```
$$ ******************************************************
$$ *              M A C H I N  S E C T I O N             *
$$ ******************************************************

CIMFIL/ON,MACHIN                                    $$ CATCH THE MACHIN COMMANDS
 DMY = POSTF(13)                                    $$ EXECUTE THE CURRENT CL RECORD
 DMY = POSTF(26,2,(ICODEF(MACHIN)),0)               $$ TURN OFF THIS FIL ROUTINE
 FN = TEXT/PART                                     $$ GET THE FILE NAME
 SLASH = TEXT/'/'                                   $$ DEFAULT TEXT STRING
 DOT = TEXT/'.'                                     $$ DEFAULT TEXT STRING
 EXT = TEXT/'PPR'                                   $$ NEW FILE EXTENSION
 NOSL) CONTIN                                       $$ JUMPTO LABEL
 SLOC = INDXF(FN,SLASH)                             $$ FIND THE SLASH
 IF(SLOC.GT.0)THEN                                  $$ FOUND A SLASH
  NCH = CANF(FN,1)                                  $$ GET THE NUMBER OF CHARACTERS IN FN
  FN = TEXT/RANGE,FN,(SLOC+1),NCH                           $$ REMOVE THE DIRECTORY NAME FROM FN
  JUMPTO/NOSL                                       $$ CHECK AGAIN FOR SLASH
 ENDIF
 DLOC = INDXF(FN,DOT)                               $$ FIND THE DOT
 IF(DLOC.GT.0)THEN                                  $$ FOUND A DOT
  NCH = CANF(FN,1)                                  $$ GET THE NUMBER OF CHARACTERS IN FN
  FN = TEXT/RANGE,FN,1,(DLOC)                       $$ REMOVE THE EXTENSION FROM FN
 ENDIF
 FN = TEXT/FN,EXT                                   $$ PPRINT FILE NAME
 FEX = FILEF(1,7,FN)                                $$ SEE IF FILE EXIST
 IF (FEX.EQ.1)THEN                                  $$ FILE EXIST
  DMY = FILEF(1,2,FN)                               $$ OPEN EXISTING FILE
  DMY = FILEF(1,6)                                  $$ CLOSE AND DELETE FILE
 ENDIF
 FOPEN = FILEF(1,3,FN)                              $$ OPEN NEW FILE
 IF(FOPEN.EQ.0)THEN                                 $$ FILE IS OPEN
  T1 = TEXT/' ************************************************************'
  DMY = FILEF(1,1,T1)                               $$ WRITE STRING TO FILE
  T1 = TEXT/' *         P P R I N T   F I L E            *'
  DMY = FILEF(1,1,T1)                               $$ WRITE STRING TO FILE
  T1 = TEXT/' ************************************************************'
  DMY = FILEF(1,1,T1)                               $$ WRITE STRING TO FILE
 ENDIF
CIMFIL/OFF

$$ ******************************************************
$$ *              P P R I N T   S E C T I O N           *
$$ ******************************************************

CIMFIL/ON,PPRINT                                    $$ CATCH THE PPRINT COMMANDS
 PPTXT = TEXT/CLW                                   $$ GET THE PPRINT TEXT FROM CL RECORD
 DMY = POSTF(13)                                    $$ EXECUTE THE CURRENT CL RECORD
 IF(FOPEN.EQ.0)THEN                                 $$ FILE IS OPEN
  DMY = FILEF(1,1,PPTXT)                            $$ WRITE STRING TO FILE
 ENDIF
CIMFIL/OFF

$$ ******************************************************
$$ *              E N D  O F  F I L  F I L E            *
$$ ******************************************************

FINI
```

6.10 FIL Example 10: How to read ahead in the CL file.

```
$$ ********************************************************
$$ *            SAMPLE 9 FIL FILE FOR TRAINING             *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *          A B O U T  T H I S  F I L  F I L E          *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO READ THE ENTIRE CL FILE
$$ AMKE A TOOL LIST AND AUTOMATICALLY OUTPUT A SELCTL AFTER EACH TOOL
$$ CHANGE OPERATION.
$$ THREE FIL ROUTINES ARE USED AS FOLLOWS
$$ MACHIN SECTION TO MAKE THE TOOL LIST
$$ LOADTL SECTION TO OUTPUT THE LOADTL / SELCTL
$$ SELCTL TO THROE AWAY THE SELCTL

$$ ********************************************************
$$ *       G L O B A L  V A R I A B L E  S E C T I O N     *
$$ ********************************************************

 REDEF/ON                               $$ ALLOW SYMBOL REDEFINITION
 LTSW = 0                               $$ DEFAULT LOADTL SWITCH
 CNT = 0                                $$ DEFAULT COUNTER
 MAXTLS = 0                             $$ MAX TOOL VARIABLE
 RESERV/TLS,100                         $$ RESERVE THE ARRAY TO STORE THE TOOL NUMBERS

$$ ********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N         *
$$ ********************************************************

 TWARN = MACRO/
   $$ THIS MACRO GETS THE TOTAL
   $$ NUMBER OF POST WARNINGS
   $$ FROM THE POST AND INCREMENTS
   $$  IT BY ONE AND RESETS THE POSTS
   $$ WITH THE NEW VALUE.
   TW = POSTF(1,1,1932)                 $$ GET THE TOTAL # OF POST WARNINGS
   TW = TW + 1                          $$ ADD 1
   DMY = POSTF(2,1,1932,TW)             $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC
```

```
$$ ******************************************************
$$ *              M A C H I N  S E C T I O N             *
$$ ******************************************************

CIMFIL/ON,MACHIN                          $$ CATCH THE MACHIN COMMANDS
 DMY = POSTF(13)                          $$ EXECUTE THE CURRENT CL RECORD
 DMY = POSTF(26,2,(ICODEF(MACHIN)),0)     $$ TURN OFF THIS FIL ROUTINE
 CLNUM = POSTF(7,1)+1                     $$ SAVE THE CURRENT CL POINTER
 $$ START TO READ THE ENTIRE CL FILE NOW
 DO/REDO,N=1,1000000,1                    $$ DO IT ALOT
   DMY = POSTF(14)                        $$ GET THE NEXT CL RECORD
   CLT = POSTF(7,2)                       $$ GET THE 2ND CL WORD, TYPE
   CLS = POSTF(7,3)                             $$ GET THE 3RD CL WORD, SUB-TYPE
   IF(CLT .EQ. 14000)THEN                 $$ FOUND THE FINI
    MAXTLS = CNT                          $$ SET THE MAX NUMBER OF TOOLS
    CNT = 0                               $$ RESET THE COUNTER
    DMY = POSTF(15,CLNUM)                 $$ REWIND THE CL FILE
    JUMPTO/DONE                           $$ GET OUT
   ENDIF
   IF(CLT .EQ. 2000 .AND. CLS .EQ. (ICODEF(LOADTL)))THEN          $$ FOUND A LOADTL
    CNT = CNT + 1                         $$ COUNT THE NUMBER OF LOADTLS
    TLS(CNT) = POSTF(7,4)                 $$ STORE THE TOOL NUMBER IN THE ARRAY TLS
   ENDIF
  REDO) CONTIN                            $$ END OF DO LOOP
  DONE) CONTIN                            $$ ESCAPE ROUTE
  $$ GOT THE TOOL LIST
CIMFIL/OFF

$$ ******************************************************
$$ *              L O A D T L   S E C T I O N           *
$$ ******************************************************

CIMFIL/ON,LOADTL                          $$ CATCH THE LOADTL COMMANDS
 DMY = POSTF(13)                          $$ EXECUTE THE CURRENT CL RECORD
 CNT = CNT + 1                            $$ COUNT THE LOADTL
 IF(CNT.GE.MAXTLS)THEN                    $$ THIS IS THE LAST TOOL
  SELCTL/TLS(1)                           $$ PRESET THE FIRST TOOL
 ELSE
  SELCTL/TLS(CNT+1)                       $$ PRESET THE NEXT TOOL
 ENDIF
CIMFIL/OFF

$$ ******************************************************
$$ *              S E L C T L   S E C T I O N           *
$$ ******************************************************

CIMFIL/ON,SELCTL                          $$ CATCH THE SELCTL COMMANDS
 $$ THROW AWAY THE SELCTLS, LOADTL TAKES CARE OF THEM
CIMFIL/OFF

$$ ******************************************************
$$ *              E N D  O F  F I L  F I L E            *
$$ ******************************************************

FINI
```

6.11 FIL Example 11: How to output data on the first motion after a command.

```
$$ ********************************************************
$$ *            SAMPLE 10 FIL FILE FOR TRAINING           *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *           A B O U T  T H I S  F I L  F I L E          *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO OUTPUT AN H CODE ON THE
$$ FIRST GOTO AFTER EACH LOADTL.
$$ THREE FIL ROUTINES ARE USED AS FOLLOWS
$$ MACHIN SECTION TO TURN OFF THE GOTO SECTION BY DEFAULT
$$ LOADTL SECTION TO SAVE THE CURRENT H OFFSET NUMBER
$$ GOTO SECTION (5,5) TO OUTPUT THE H CODE


$$ ********************************************************
$$ *       G L O B A L  V A R I A B L E  S E C T I O N    *
$$ ********************************************************

 REDEF/ON                               $$ ALLOW SYMBOL REDEFINITION
 LTSW = 0                               $$ DEFAULT LOADTL SWITCH
 CNT = 0                                $$ DEFAULT COUNTER

$$ ********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N         *
$$ ********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS
  $$ FROM THE POST AND INCREMENTS
  $$  IT BY ONE AND RESETS THE POSTS
  $$ WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                  $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                           $$ ADD 1
  DMY = POSTF(2,1,1932,TW)              $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ ********************************************************
$$ *              M A C H I N  S E C T I O N              *
$$ ********************************************************

CIMFIL/ON,MACHIN                        $$ CATCH THE MACHIN COMMANDS
 DMY = POSTF(13)                        $$ EXECUTE THE CURRENT CL RECORD
 DMY = POSTF(26,2,(ICODEF(MACHIN)),0)   $$ TURN OFF THIS FIL ROUTINE
 DMY = POSTF(26,5,5,0)                  $$ TURN OFF CIMFIL/ON,5,5 FIL ROUTINE
CIMFIL/OFF
```

```
$$ ******************************************************
$$ *              L O A D T L   S E C T I O N           *
$$ ******************************************************

CIMFIL/ON,LOADTL                          $$ CATCH THE LOADTL COMMANDS
 DMY = POSTF(13)                          $$ EXECUTE THE CURRENT CL RECORD
 OFVAL = POSTF(1,1,1942)                  $$ GET THE VALUE FROM INTCOM 1942 (OFFSET LOCATION)
 LTSW = 1                                 $$ SET THE LOADTL SWITCH
 DMY = POSTF(26,5,5,1)                    $$ TURN ON THE CIMFIL/ON,5,5 ROUTINE
CIMFIL/OFF

$$ ******************************************************
$$ *           G O T O   P O I N T   S E C T I O N      *
$$ ******************************************************

CIMFIL/ON,5,5                             $$ CATCH THE GOTO POINTS
 DMY = POSTF(20)                          $$ SAVE THE CURRENT CL RECORD
 IF(LTSW .EQ.1)THEN                       $$ FIRST MOVE AFTER LOADTL
  POSTN/IN,8,OFVAL,NEXT                   $$ OUTPUT Hofval WITH NEXT BLOCK
  LTSW  = 0                               $$ RESET THE LOADTL SWITCH
 ENDIF
 DMY = POSTF(21)                          $$ RELOAD THE SAVED CL RECORD
 DMY = POSTF(13)                          $$ EXECUTE THE CURRENT CL RECORD
 DMY = POSTF(26,5,5,0)                    $$ TURN OFF THIS CIMFIL/ON,5,5 ROUTINE
CIMFIL/OFF

$$ ******************************************************
$$ *            E N D   O F   F I L   F I L E           *
$$ ******************************************************

FINI
```

6.12 FIL Example 12: How to change post settings based on other post commands.

```
$$ ********************************************************
$$ *            SAMPLE 11 FIL FILE FOR TRAINING           *
$$ ********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ ********************************************************
$$ *          A B O U T  T H I S  F I L  F I L E          *
$$ ********************************************************

$$ THIS IS A SAMPLE FIL FILE TO SHOW HOW TO USE THE FROM VALUES FOR
$$ THE GOHOME COMMAND AND TO OUTPUT A G92 BLOCK TO THE FROM POINT
$$ ONE FIL ROUTINE IS USED AS FOLLOWS:  FROM (5,3) SECTION

$$ ********************************************************
$$ *       G L O B A L  V A R I A B L E  S E C T I O N    *
$$ ********************************************************

 REDEF/ON                                 $$ ALLOW SYMBOL REDEFINITION
 LTSW = 0                                 $$ DEFAULT LOADTL SWITCH

$$ ********************************************************
$$ *        G L O B A L  M A C R O  S E C T I O N         *
$$ ********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS
  $$ FROM THE POST AND INCREMENTS
  $$  IT BY ONE AND RESETS THE POSTS
  $$ WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                    $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                             $$ ADD 1
  DMY = POSTF(2,1,1932,TW)                $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ ********************************************************
$$ *              F R O M  S E C T I O N                  *
$$ ********************************************************

CIMFIL/ON,5,3                             $$ CATCH THE FROM COMMANDS
 FRX = POSTF(7,6)                         $$ GET THE 6TH CL WORD , FROM X VALUE
 FRY = POSTF(7,7)                         $$ GET THE 7TH CL WORD , FROM Y VALUE
 FRZ = POSTF(7,8)                         $$ GET THE 8TH CL WORD , FROM Z VALUE
 DMY = POSTF(2,1,1867,1)                  $$ POST SIMULATION MODE ON
 DMY = POSTF(13)                          $$ EXECUTE THE CURRENT CL RECORD
 DMY = POSTF(2,1,1867,0)                  $$ POST SIMULATION MODE OFF
 SET/START,XAXIS,FRX,YAXIS,FRY,ZAXIS,FRZ  $$ OUTPUT G92 BLOCK
 DMY = POSTF(2,3,130,FRX)                 $$ RESET GOHOME X
 DMY = POSTF(2,3,131,FRY)                 $$ RESET GOHOME Y
 DMY = POSTF(2,3,132,FRZ)                 $$ RESET GOHOME Z
CIMFIL/OFF

$$ ********************************************************
$$ *              E N D  O F  F I L  F I L E              *
$$ ********************************************************

FINI
```

6.13 FIL Example 13: How to read the PARTNO to retrieve information.

```
$$ *********************************************************
$$ *              SAMPLE 12 FIL FILE FOR TRAINING          *
$$ *********************************************************

$$          PRINT/OFF,IN                    $$ USE FOR PRODUCTION
            PRINT/ON                        $$ USE FOR TESTING
            DMY = POSTF(24,1)               $$ USE FOR TESTING

$$ *********************************************************
$$ *        G L O B A L   V A R I A B L E   S E C T I O N  *
$$ *********************************************************

REDEF/ON                        $$ allow symbol redefinition

$$ *********************************************************
$$ *         G L O B A L   M A C R O   S E C T I O N       *
$$ *********************************************************

 TWARN = MACRO/
  $$ THIS MACRO GETS THE TOTAL
  $$ NUMBER OF POST WARNINGS
  $$ FROM THE POST AND INCREMENTS
  $$  IT BY ONE AND RESETS THE POSTS
  $$ WITH THE NEW VALUE.
  TW = POSTF(1,1,1932)                  $$ GET THE TOTAL # OF POST WARNINGS
  TW = TW + 1                           $$ ADD 1
  DMY = POSTF(2,1,1932,TW)              $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ *********************************************************
$$ *                  P A R T N O   S E C T I O N          *
$$ *********************************************************
CIMFIL/ON,PARTNO                        $$ SPOST standard PARTNO sequence ...
LOOPST                                  $$ ================================
 PLETER = TEXT/'O'                      $$ PLETER = string required for program
                                        $$              number block identifier
 PARLEN = 8                             $$ PARLEN = REQUIRED # of chars for
                                        $$              program number
 MINMIN = 90000000                      $$ MINMIN = lowest acceptable program #
 MAXMAX = 99999999                      $$ MAXMAX = largest    "         "    #
 BLANKS = TEXT/REPEAT,66,' '            $$ BLANKS = string of blanks (EMPTY)
 CONCOD = 0                             $$ CONCOD = 0 indicates good PARTNO

 $$ .....  CHECK PARTNO FOR ALL BLANKS  ..............................

 PARSTR = TEXT/CLW                      $$ save the text from the CL record
 PARSPN) CONTIN                         $$ branch to here from prompt section
 CONCOD = CMPRF(PARSTR,BLANKS)          $$ check for empty (66 blanks)
 IF (CONCOD .EQ. 1) THEN                $$ CONCOD = 1; empty PARTNO string
  ASK1TX = TEXT/'PROGRAM NUMBER FIELD EMPTY, ENTER PROGRAM NUMBER'
  JUMPTO/ASK1
 ENDIF                                  $$ PARTNO wasn't empty, continue

 $$ .....  CHECK PARTNO STRING FOR SUFFICIENT LENGTH ...................

 PARSTR = TEXT/OMIT,PARSTR,3                    $$ omit ALL blanks
 PGMLEN = CANF(PARSTR,1)                $$ get length of remaining string
 IF (PGMLEN .LT. PARLEN) THEN
  CONCOD = 2                            $$ CONCOD = 2; wrong number of chars
 ELSE
  PARSTR = TEXT/RANGE,PARSTR,1,PARLEN   $$ reduce string to correct length
 ENDIF
```

```
 IF (CONCOD .NE. 0) THEN
   ASK2)CONTIN                                  $$ error trap for blank answer
   ASK1TX = TEXT/$
  'PROGRAM NUMBER MUST BE ',PARLEN,' CHARS IN LENGTH, RE-ENTER'
   JUMPTO/ASK1
 ENDIF

 $$ .....  CHECK PARTNO STRING FOR NON NUMERIC CHARACTERS  .....

 BAD = 0
 DO/VFY,KK = 1,PARLEN                         $$ check PARSTR - only numbers allowed
    1CHR = TEXT/RANGE,PARSTR,KK,KK            $$ look at each individual character
    1VAL = ICHARF(1CHR)
    IF (1VAL .LT. 48 .OR. $
        1VAL .GT. 57) BAD = BAD+1             $$ if BAD not zero, non-numeric found!
    IF (BAD .NE. 0) KK = PARLEN+1             $$ get out of loop on 1st bad character
 VFY) CONTIN                                  $$ end of DO loop ...
 IF (BAD .NE. 0) CONCOD = 3                   $$ CONCOD = 3; non numeric char found!

 IF (CONCOD .NE. 0) THEN
   ASK1TX = TEXT/'NON-NUMERIC VALUES NOT ALLOWED, RE-ENTER'
   JUMPTO/ASK1
 ENDIF

 $$ .....  CHECK PARTNO STRING TO BE WITHIN ACCEPTABLE RANGE  .....

 PARNUM = SCALF(PARSTR)                       $$ convert string into scalar equiv.
 IF (PARNUM .LT. MINMIN  .OR. $
     PARNUM .GT. MAXMAX) CONCOD = 4           $$ CONCOD = 4; value not in reqd range
 IF (CONCOD .NE. 0) THEN
   ASK1TX = TEXT/$
   'PROGRAM NUMBER MUST BE BETWEEN ',MINMIN,' AND ',MAXMAX,' ,RE-ENTER'

 $$ ..........................................................................

   ASK1) CONTIN                                        $$ didn't like something, prompt
     XX = FILEF(0,1,ASK1TX)                   $$ write prompt to screen
     PARSTR = TEXT/READ,0                     $$ get the users answer
     NEWLEN = CANF(PARSTR,1)                  $$ get length of new answer
     IF (NEWLEN .LT. 1) JUMPTO/ASK2
     BLANKS = TEXT/REPEAT,NEWLEN,' '          $$ make new "blanks" correct length
     JUMPTO/PARSPN                            $$ goto top and test this string
 ENDIF                                        $$ if we got here, PARTNO is OK!!
 PARSTR = TEXT/PLETER,PARSTR,'!'              $$ PARSTR is string ready to INSERT
                                              $$ AFTER processing MACHIN stmt.
 LOOPND
 PARTNO/PARSTR
CIMFIL/OFF

$$ ********************************************************
$$ *              M A C H I N   S E C T I O N             *
$$ ********************************************************

CIMFIL/ON,MACHIN
  XX = POSTF(13)                              $$ execute the current CL record
  INSERT/PARSTR                               $$ output the program number from PARTNO
CIMFIL/OFF

$$ ********************************************************
$$ *                E N D   O F   F I L   F I L E        *
$$ ********************************************************

FINI
```

6.14 FIL Example 14: How to catch the CLEARP command.

```
$$ *********************************************************
$$ *              SAMPLE 13 FIL FILE FOR TRAINING          *
$$ *********************************************************

$$          PRINT/OFF,IN                        $$ USE FOR PRODUCTION
            PRINT/ON                            $$ USE FOR TESTING
            DMY = POSTF(24,1)                   $$ USE FOR TESTING

$$ *********************************************************
$$ *       G L O B A L   V A R I A B L E   S E C T I O N   *
$$ *********************************************************

REDEF/ON                        $$ allow symbol redefinition

$$ *********************************************************
$$ *        G L O B A L   M A C R O   S E C T I O N        *
$$ *********************************************************

 TWARN = MACRO/
   $$ THIS MACRO GETS THE TOTAL
   $$ NUMBER OF POST WARNINGS
   $$ FROM THE POST AND INCREMENTS
   $$  IT BY ONE AND RESETS THE POSTS
   $$ WITH THE NEW VALUE.
   TW = POSTF(1,1,1932)                         $$ GET THE TOTAL # OF POST WARNINGS
   TW = TW + 1                                  $$ ADD 1
   DMY = POSTF(2,1,1932,TW)                     $$ RESET THE TOTAL # OF POST WARNINGS
 TERMAC

$$ *************************************************************
$$ *                   C L E A R P   S E C T I O N             *
$$ *************************************************************

CIMFIL/ON,CLEARP                                $$ CATCH THE CLEARP COMMANDS
  CLV = POSTF(7,4)                              $$ GET THE 4TH CL WORD
  CLT = POSTF(6,4)                              $$ GET THE 4TH CL  WORD TYPE
  IF(CLT .EQ. 0 .AND. CLV .EQ. (ICODEF(OFF)))THEN
    TCZ = POSTF(1,3,606)                        $$ TOOL CHANGE Z
    TRZ = POSTF(1,3,288)                        $$ FIXED TRANS Z
    CLV = (TCZ + (TRZ))                         $$ REMOVE THE Z TRANSLATION
    DMY = POSTF(10,5,CLV)                       $$ LOAD THE NEW CLEARP VALUE
  ELSE
    CLV = POSTF(7,5)                            $$ GET THE 5TH CL WORD
    CLT = POSTF(6,5)                            $$ GET THE 5TH CL WORD TYPE
    IF(CLT .EQ. 1)THEN
      TRZ = POSTF(1,3,288)                      $$ FIXED TRANS Z
      CLV = (CLV + (TRZ))                       $$ REMOVE THE Z TRANSLATION
      DMY = POSTF(10,5,CLV)                     $$ LOAD THE NEW CLEARP VALUE
    ENDIF
  ENDIF
  DMY = POSTF(13)                               $$ EXECUTE CURRENT CL RECORD
CIMFIL/OFF

$$ *********************************************************
$$ *                 E N D   O F   F I L   F I L E         *
$$ *********************************************************

FINI
```

6.15 FIL Example 15: How to Examine a CL Record.

This macro gives you all sorts of information about the CL record. In this example it is used in two CIMFIL sections.

```
1.  CLDUMP=MACRO
2.      DMY=POSTF(20)
3.      MXCL=POSTF(5)
4.      CLSTRG=TEXT/'==='
5.      WRD=POSTF(7,3)
6.      INUM=TEXT/CONVI,WRD,5
7.      INUM=TEXT/OMIT,INUM,3
8.      CLSTRG=TEXT/CLSTRG,' ',INUM
9.      DO/CLD100,I=4,MXCL,1
10.       TYPCOD=POSTF(6,I)
11.       CASE/TYPCOD
12.        WHEN/0
13.              WRD=POSTF(7,8)
14.              WRDTXT=TEXT/CONVI,WRD,5
15.        WHEN/1
16.              WRDTXT=TEXT/CONVF,WRD,9,4,0,0,3
17.        WHEN/0,1
18.              WRDTXT=TEXT/OMIT,WRDTXT,3
19.              CLSTRG=TEXT/CLSTRG,' ',WRDTXT
20.        WHEN/2
21.              PPRINT/'ERROR !!!! TEXT VARIABLE'
22.        WHEN/OTHERS
23.              PPRINT/'ERROR !!!! ILLEGAL TYPE CODE'
24.       ENDCAS
25.     CLD100)CONTIN
26.     PPRINT/CLSTRG
27.     DMY=POSTF(21)
28. TERMAC
29. CIMFIL/ON,2
30.     WD=POSTF(7,3)
31.     IF(WD.NE.1044.AND.WD.NE.1045.AND.WD.NE.1046$
          .AND.WD.NE.1015)THEN
32.     CALL/CLDUMP
33.     DMY=POSTF(13)
34. CIMFIL/OFF
35. CIMFIL/ON,5
36.     CALL/CLDUMP
37.     DMY=POSTF(13)
38. CIMFIL/OFF
39. FINI
```

Here is a line-by-line explanation of the preceding code.

1 Begin macro
2 Save CL record
3 Get CL record word count
4 Set up text string variable
5 Get value at location 3
6 Convert value to integer
7 Remove all blanks
8 Add a space between numbers
9 Begin DO loop
10 Find the word type
11 Read the word type
12 Find vocabulary word
13 Get value
14 Convert to integer
15 Find scalar
16 Format scalar
17 Find vocabulary word and scalar
18 Remove all blanks
19 Add a space between numbers
20 Find text string
21 Print debugging statement
22 Error trap
23 Print debugging statement
24 End CASE statement
25 End DO loop
26 Print text string
27 Restore saved CL record
28 End macro
29 Find all Post (Type 2000) records
30 Get value
31 Ignore PPRINT, PARTNO, INSERT, MACHIN statements
32 Call macro
33 Process results
34 End CIMFIL
35 Trap all motion statements
36 Call macro
37 Process results
38 End CIMFIL
39 End FIL file

6.16 FIL Example 16: How to Introduce a New Minor Word to an Existing Command

We want to add a new minor word, **AUTOX**, to the **SET** statement in our post.

The first thing we are going to do is tell the G-Post system that **AUTOX** is a valid vocabulary word and has an value. This is done with the **PPWORD** statement. The **PPWORD** statement needs to assign the value/or integer code to the word. The available range of integer codes are between 3101 and 4095 for a major or minor word. For this example we choose 3101 since it is the first unused integer code.

We will add the following information to our CL file.

PPWORD/AUTOX,3101
SET/AUTOX

Note: The **PPWORD** statement can also be added to your vocabulary table file.

In the CL file we noticed that there were several **SET** statements. We will use an **IF-THEN-ELSE** logical statement to determine the ones we do not want so they will be process normally.

Example:

```
CIMFIL/ON,SET                                  $$ Trap all SET statements
  RSLT=POSTF(20)                               $$ Saves current CL record
  TYPE=POSTF(6,4)                              $$ Look at location 4
  IF (TYPE .EQ. 0) THEN                        $$ If a minor word (Type 0),
                                               $$  process the next line
   WORD=POSTF(7,4)                             $$ Return the value
    IF (WORD .EQ. (ICODEF(AUTOX))) THEN        $$ If  3101, process the next line
     PPRINT HERE'S AUTOX                       $$ Print the statement
    ELSE                                       $$ If not 3101, process next line
     RSLT=POSTF(13)                            $$ Process the record
   ENDIF                                       $$ End the search
 ELSE                                          $$ If the 4th location isn't
                                               $$  a minor word, process the next line
   RSLT=POSTF(13)                              $$ Process the record
 ENDIF                                         $$ End the value search
CIMFIL/OFF                                     $$ Stop FIL
```

6.17 FIL Example 17: How to Combine Codes.

This example shows you how to combine the **TURRET** direction M-code onto the block containing the T-code. The G-Post default is to output a M-code for direction in the block preceding the T-code block.

The G-Post produces the following code:

N050M13$
N055G04T0101$
N060G92X10Z03$

The machine requires the following code:

N050G04T0101M13$
N055G92X10Z03$

Example CL File:

1. PARTNO TEST FOR TURRET
2. FROM/10,10
3. SPINDL/300,RPM
4. TURRET/1,1,0,0
5. RAPID
6. GOTO/0,0
7. GOTO/0,-.100
8. RAPID
9. GOTO/0,7
10. TURRET/4,4,0,0,CLW
11. RAPID
12. GOTO/0,4.5
13. RAPID
14. GOTO/10,10
15. TURRET/5,5,0,0,CCLW
16. GOTO/0,10
17. RAPID
18. GOTO/10,10
19. END
20. FINI

Example FIL File:

```
 1  PARTNO TURRET FIL FILE EXAMPLE
 2  $$ PPRINT/OFF,IN                          $$ Use for production
 3  $$ PRINT/ON                               $$ Use for debug
 4  REDEF/ON
 5  MCL=TEXT/'M13'
 6  MCCL=TEXT/'M14'
 7  EOBCHR = TEXT/'$'
 8  BLNK = TEXT/' '
 9  ENDSTG = TEXT/'ERROR$EOF'
10  CIMFIL/ON,TURRET
11     I1 = POSTF(20)
12     NW = POSTF(5)
13      IF (NW .LT. 8) THEN
14        I1 = POSTF(13)
15      ELSE
16        OPSAV = POSTF(1,1,1511)
17        I1 = POSTF(2,1,1511,(ICODEF(OFF)))
18        $$ POSTF(24,1)                        $$ Uncomment this for debug
19        I1 = POSTF(19)
20        SEQNO/OFF
21        I1 = POSTF(25,1)
22        I1 = POSTF(21)
23        I1 = POSTF(13)
24        I1 = POSTF(19)
25        I1 = POSTF(25,0)
26        SEQNO/ON
27        DO/TURLOP,CT=1,10,1
28         TURTX = TEXT/READ,PUNCH
29         TURREX = CMPRF(TURTX,ENDSTG)
30          IF (TURREX .NE. 0) JUMPTO/TUREND
31          FOND13 = INDXF(TURTX,MCL)
32          FOND1INDXF(TURTX,MCCL)
33          IF (FOND13.NE.0 .OR. FOND14.NE.0) THEN
34           TMP = TEXT/READ,PUNCH
35           TMP = TEXT/MODIFY,TMP,EOBCHR,BLNK,0
36           TURTX = TEXT/TMP,TURTX
37         ENDIF
38         INSERT/TURTX
39       TURLOP) CONTIN
40       TUREND) CONTIN
41        I1 = POSTF(2,1,1511,OPSAV) <$IPOSTF(2)>
42    ENDIF
43  CIMFIL/OFF
44  FINI
```

And here is an explanation of some of the FIL code. The number represents the line number of the FIL file command that is being explained.

4	Turn on REDEF so variables can be reassigned
5	String for CLW index code
6	String for CCLW index code
7	String for EOB character
8	String for blank character
9	This is the string returned when an end-of-file is encountered
10	Trap all TURRET commands
11	Save TURRET CL record
12	Get pointer for last entry on statement. Note: First minor entry (right of /) is word 4. TURRET/N,N,N,N,CLW would return a value of 8 in NW.
13	Trap only TURRET/,[CLW or CCLW]
14	Don't operate on this command!
16	Save contents of OPSKIP switch
17	Set OPSKIP to OFF. Don't let any /'s appear in re-directed output
19	Dump any pending output
20	Turn off seq numbers for output to be redirected. This will overwrite the TURRET CL record.POSTF(20) saved it
21	Redirect print and punch to auxiliary files
22	Restore TURRET CL record
23	Execute current TURRET cmd; output to aux files
24	Dump any output left pending by TURRET
25	Reset print & punch to primary files. Set up loop to go thru all output in aux punch file. Find M13 or M14 and merge onto next block.
26	Reinstate sequence numbers in output
27	Scan through a maximum of 10 blocks
29	Check for end-of-file condition
30	If end of file, exit loop
31	M13 or M14 located. Get rid of EOB char in T-code block and add block w/ M-dir code (MLOC) to block w/ T-code (MLOC+1). (This will only work if T block immediately follows M-dir block)
34	Get block following M-code and discard EOB character
36	Merge M-code block onto T-code block
38	Send in modified or same text to post
40	Exit TURRET modification
41	Set OPSKIP switch back to saved value
44	FINI statement is optional, and not required

6.18 FIL Example 18: How to Customize the COOLNT Command.

This example shows how to introduce a custom **COOLNT** command format. This machine has the normal coolant on/off codes plus M12, 37, and 38. This example adds a new word **FLUSH** to the CL vocabulary and uses it as a minor parameter of **COOLNT**. Remember, the **PPWORD** statement adds the word and its associated integer code. This statement must be in any CL file that uses **FLUSH** and/or the vocabulary table file.

You should appoint a coordinator at your site to administer the specific WORD,INTEGER CODE additions you make with **PPWORD**. This particular post (UNCX01,3) expects **FLUSH** to have an integer code of 3117. Without a site vocabulary administrator, another post could establish **FLUSH** as a different integer code or a different word as having integer code 317.

Warning: Either of these conditions is potentially dangerous.

This FIL file establishes the following syntax for MACHIN/UNCX01,3:

```
COOLNT/THRU        = M07
       /FLOOD      = M08
       /OFF        = M09
       /HIGH       = M12
COOLNT/FLUSH       = M37
COOLNT/FLUSH,OFF   = M38
```

Any other **COOLNT** command format will be diagnosed as an error. However, the appropriate output will still be generated if it's possible
.
When the Post encounters a **COOLNT/...,NEXT** statement, it will try to put the M-code on the next block.

Example CL File:

```
PARTNO TEST FIL FILE FOR COOLNT
PPWORD/FLUSH,3117                    $$ Used by FIL
FROM/0,0,0
FEDRAT/20,IPM
SPINDL/300,RPM
COOLNT/FLOOD
GOTO/1,1,1
COOLNT/OFF
GOTO/2,2,2
COOLNT/HIGH
COOLNT/THRU
GOTO/0,0,0
COOLNT/FLUSH,NEXT
GOTO/1,1,1
COOLNT/FLUSH,OFF
GOTO/0,0,0
COOLNT/OFF
GOTO/3,3,3
COOLNT/FLUSH,ON
GOTO/0,0,0
```

```
COOLNT/FLOOD,ON
END
FINI
```

The G-Post allows you to use **DISPLY** to specify that all **PPRINT** statements be output to the punch file. This FIL file uses **PPRINT** to produce error messages in the LIST file. To keep these messages from appearing in the punch file, you must save the DISPLY switch, then turn it off. Remember to reinstate the DISPLY switch at the end of the FIL file. See the G-Post User's Guide for rules concerning the **PPRINT** statement.

Example FIL Code:

```
          PARTNO COOLNT FIL FILE EXAMPLE UNCX01
1         $$  PRINT/OFF,IN                    $$ Uncomment this for production
2         $$  PRINT/ON                        $$ Uncomment this to debug
3         PPWORD/FLUSH,317
4         CIMFIL/ON,COOLNT
5           $$  POSTF(24,1)                   $$ Uncomment this for debug
6           CSAV=POSTF(1,1,484)
7           XX=POSTF(2,1,484,(ICODEF(OFF)))
8           NW=POSTF(5)
9           IF (NW .LT. 4) THEN
10            PPRINT/'**ERROR**NO MINOR WORD ON COOLNT'$
              ' CMD: CMD IGNORED'
11            JUMPTO/COOL99
12          ELSE
13            CONTIN
14          ENDIF
15          IWRD=POSTF(7,4)
16          CASE/IWRD
17           WHEN/ICODEF(THRU)
18             COLCOD=07
19           WHEN/ICODEF(FLOOD)
20             COLCOD=08
21           WHEN/ICODEF(OFF)
22             COLCOD=09
23           WHEN/ICODEF(HIGH)
24             COLCOD=12
25           WHEN/ICODEF(FLUSH)
26             COLCOD=37
27             IF (NW .LT. 5) JUMPTO/COOL1
28             IWRD=POSTF(7,5)
29             IF (IWRD .EQ. (ICODEF(OFF))) THEN
30               COLCOD=38
31             ELSE
32               IF (IWRD .EQ. (ICODEF(NEXT)))JUMPTO/COOL1
33               PPRINT/'**ERROR** COOLNT/FLUSH,W; W=OFF/NEXT'
34               JUMPTO/COOL99
35             ENDIF
36           WHEN/OTHERS
37             PPRINT/'**ERROR**1ST MNR WRD ON COOLNT'$
                        ,' IN ERROR:CMD IGNORED'
38             JUMPTO/COOL99
39          ENDCAS
40          COOL1) CONTIN
41          IWRD=POSTF(7,NW)
```

```
42          IF (IWRD .EQ. (ICODEF(NEXT))) THEN
43            AUXFUN/COLCOD,NEXT
44          ELSE
45            AUXFUN/COLCOD
46          ENDIF
47          COOL99) CONTIN
48          XX=POSTF(2,1,484,CSAV)
49  CIMFIL/OFF
50  FINI
```

And here is an explanation of some of the FIL code. The number represents the line number of the FIL file command that is being explained.

```
 3  Establish IC 317 for FLUSH
 4  Trap all COOLNT commands
 6  Save the switch contents in CSAV
 7  Set the switch to OFF
 8  See how many words are in command; 1st minor word (right of slash) is word 4 in CL record. NW=4
    for COOLNT/OFF; NW=5 for /FLUSH,OFF
 9  If no minor word on COOLNT, error
12  Some type of minor entry is there
15  Get IC for first word right of the slash
17  IWRD=integer code of THRU
18  M-code of 07
19  IWRD=integer code of FLOOD
20  M-code of 08
21  IWRD=integer code of OFF
22  M-code of 09
23  IWRD=integer code of HIGH
24  M-code of 12
25  IWRD=integer code of FLUSH
26  M-code of 27
27  COOLNT/FLUSH-no 2nd minor word
28  IWRD now contains 2nd minor word
29  2nd minor word is OFF
36  No match, bad COOLNT command
41  IWRD now contains the last minor word. If NEXT is on the end of the COOLNT command, try to
    merge M-code onto the next block
48  Reset IDSPLY switch to the condition it had upon entry of this routine
49  End CIMFIL
```

Example of Post Processor Output:

```
TEST THE FIL FILE FOR COOLNT  (INCH)
INPUT  CLREC
N4G2X34Y34R34Z34I34J34K34B33F33S4T5D2H2M2

7     7   $ TEST THE FIL FILE FOR COOLNT
7     7   $
7     7   LEADER/   14.0
7     7   N0001 G70$
7     7   N0002 G17$
7     7   N0003 G90$
9    11    N0004 M41$
9    11    N0005 S0300 M03$
10    13    N0006 M08$
```

```
11    15   N0007 G01 X001 Y001 Z001 F02$
12    17   N0008 M09$
13    19   N0009 X002 Y002 Z002$
14    21   N0010 M12$
15    23   N0011 M07$
16    25   N0012 X0 Y0 Z0$
18    29   N0013 X001 Y001 Z001 M37$
19    31   N0014 M38$
20    33   N0015 X0 Y0 Z0$
21    35   N0016 M09$
22    37   N0017 X003 Y003 Z003$
23    39    **ERROR**COOLNT/FLUSH,W; W = OFF/NEXT
24    41   N0018 X0 Y0 Z0$
25    43   N0019 M08$
26    45   N0020 M02$
27    47   LEADER/   36.0
```

6.19 FIL Example 19: How to Swap Locations of a Minor Word and Value.

In this example, our CAD system gave us SPINDL/CLW,300. Unfortunately, the Post wants SPINDL/300,CLW. We'll use FIL to switch the position of the minor word and its value. And since there's almost always more than one way to do anything, we show you two methods.

Example Method 1:

```
CIMFIL/ON,SPINDL                $$ Trap SPINDL statements
 RSLT=POSTF(20)                 $$ Save record
 MAXWDS=POSTF(5)                $$ No. words in the CL record
 IF(MAXWDS .GT. 4)THEN          $$ If at least 4 words
  TYPE=POSTF(6,4)               $$ Get the 4th CL record type
  IF(TYPE .EQ. 0)THEN           $$ Find minor word
    WORD=POSTF(7,4)             $$ Return value in WORD
    TYPE2=POSTF(6,5)            $$ Get the 5th CL word type
    IF(TYPE2 .EQ. 1)THEN        $$ Find scalar
      SPEED=POSTF(7,5) $$ Return value in SPEED
      RSLT=POSTF(10,4,SPEED)    $$ Put SPEED in the 4th CL word location
      RSLT=POSTF(9,5,WORD)      $$ Put WORD in the 5th CL word location
    ENDIF                       $$ End scalar search
  ENDIF                         $$ End minor word search
 ENDIF                          $$ End MAXWDS search
 RSLT=POSTF(13)                 $$ Process the record
CIMFIL/OFF                      $$ End file
```

Example Method 2:

```
CIMFIL/ON,SPINDL                $$ Trap SPINDL statement
 RSLT=POSTF(20)                 $$ Save record
 MAXWDS=POSTF(5)                $$ No. words in CL record
 IF(MAXWDS .GT. 4)THEN          $$ If at least 4 words
  TYPE=POSTF(6,4)               $$ Get the 4th CL word type
  IF(TYPE .EQ. 0)THEN           $$ Find minor word
    WORD=POSTF(7,4)             $$ Return value in WORD
    SPEED=POSTF(7,5)            $$ Return value in SPEED
    SPINDL/SPEED,CONVI,WORD  $$ Convert IC to word
  ELSE
    POSTF(13)                   $$ OK as is
  ENDIF                         $$ End minor word search
 ENDIF                          $$ End MAXWDS search
CIMFIL/OFF                      $$ End file
```

6.20 FIL Example 20: The MAD Macros.

The MAD (make address) Macros, MAD1 and MAD2 are used in a FIL file to set letter address aliases. Use MAD1 to set the alias and MAD2 to reset the letter address back to its original letter address.

Example MAD1:

```
$$ MAD1 IS A MACRO TO CHANGE LETTER ADDRESS TO STRING
$$ ADR IS ADDRESS TO BE SET; A=1,B=2 ETC
$$ STR IS THE ALIAS STRING; TB=TEXT/'R101='
$$ updated ; del ;  18-nov-96

MAD1=MACRO/ADR,STR
 TMPAD1 = ADR + 64                                    $$ set the decimal value
 TMPAD2 = ADR + 64 + 8192                             $$ SET THE DECIMAL VALUE
 ADSW = 0                                             $$ ERROR SWITCH
 DO/REDOAD,N=1,26,1                                   $$ LOOK FOR ADDRESS IN JADDR
  NXTADR = POSTF(1,1,(52+N))                          $$ READ THE NEXT ADDRESS VALUE
  IF(NXTADR .EQ. TMPAD1 .OR. NXTADR .EQ. TMPAD2)THEN $$ IF ADDRESS FOUND
   L1 = N                                             $$ RESET L1
   ADRSW = 1                                          $$ SET THE ERROR SWITCH OK
   JUMPTO/DONEAD                                      $$ GET OUT
  ENDIF
 REDOAD) CONTIN
 DONEAD) CONTIN
 IF(ADRSW .EQ. 0)THEN
  TW = POSTF(1,1,1932)                          $$ GET THE TOTAL # OF WARNINGS
  TW = TW + 1                                          $$ ADD 1 WARNING
  DMY = POSTF(2,1,1932,1)                              $$ RESET THE TOTAL # OF WARNINGS
  PPRINT/'***WARNING, NO ADDRESS FOR MAD1 MACRO'  $$ OUTPUT A MESSAGE
  JUMPTO/L11
 ENDIF
 L2=(9-1)*26+L1+2216-1                                 $$ COMPUTE JPARAD(X,9)
 I1=POSTF(2,1,L2,1)                                    $$ SET JPARAD(X,9)
 DO/L10,N1=1,8                                  $$ ALLOW UPTO 8CHARS
  L2=(N1-1)*26+L1+2216-1                               $$ COMPUTE JPARAD(X,N)
  I1=POSTF(3,1,L2)                                     $$ SET JPARAD(X,N)=IEMPTY
  IF(N1 .GT. CANF(STR,1))JUMPTO/L10                    $$ SKIP IF OVER STRING LENGTH
  A1=ICHARF((TEXT/RANGE,STR,N1,N1))                    $$ GET CHARACTER NUMBER
  I1=POSTF(2,1,L2,A1)                                  $$ INSERT STRING(N)
 L10)CONTIN
 L11)CONTIN
TERMAC
```

Example MAD2:

```
$$ MAD2 IS A MACRO TO RESET THE LETTER ADDRESS BACK FROM A STRING
$$ ADR IS THE ADDRESS TO BE RESET; A=1,B=2 ETC
$$ UPDATED ; DEL ; 18-NOV-96

MAD2=MACRO/ADR
 TMPAD1 = ADR + 64                                   $$ set the decimal value
 TMPAD2 = ADR + 64 + 8192                            $$ SET THE DECIMAL VALUE
 ADSW = 0                                            $$ ERROR SWITCH
 DO/REDOAD,N=1,26,1                                  $$ LOOK FOR ADDRESS IN JADDR
  NXTADR = POSTF(1,1,(52+N))                         $$ READ THE NEXT ADDRESS VALUE
  IF(NXTADR.EQ.TMPAD1.OR.NXTADR.EQ.TMPAD2)THEN       $$ FOUND THE ADDRESS
   L1 = N                                            $$ RESET L1
   ADRSW = 1                                         $$ SET ERROR SWITCH 1 = NO ERROR
   JUMPTO/DONEAD                                     $$ GET OUT
  ENDIF
 REDOAD) CONTIN
 DONEAD) CONTIN
 IF(ADRSW .EQ. 0)THEN                                $$ NO ADDRESS IN JADDR
  TW = POSTF(1,1,1932)                               $$ GET THE TOTAL # OF WARNINGS
  TW = TW + 1                                        $$ ADD 1 WARNING
  DMY = POSTF(2,1,1932,1)                            $$ RESET THE TOTAL NUMBER OF
WARNINGS
  PPRINT/'***WARNING, NO ADDRESS FOR MAD2 MACRO' $$ OUTPUT MESSAGE
  JUMPTO/L11
 ENDIF
 L2=(9-1)*26+L1+2216-1                               $$ COMPUTE JPARAD(X,9)
 I1=POSTF(3,1,L2)                                    $$ SET JPARAD(X,9)=IEMPTY
 L11) CONTIN
TERMAC
```

6.21 FIL Example 21: Remove the Punch File data when an Error occurs.

This example shows you how to remove all the data from the Punch file when an Error condition is found with a FIL routine.

Note: The **REWIND/PUNCH** command causes the file pointer to be repositioned to the first line of the Punch file. Any subsequent write to the Punch file will cause a loss of all data currently in the file. Forcing the G-Post to exit, by sending it the command **FINI**, after using **REWIND/PUNCH** will cause the Punch file to be empty.

Warning: Using this example incorrectly can cause your Punch file to empty.

Example:

```
$$ SAMPLE TO KILL PUNCH FILE ON CERTAIN WARNINGS
REDEF/ON
$$PRINT/ON
$$
$$ CATCH WARNINGS
CIMFIL/ON,MACHIN
  DMY=POSTF(13)
  PLABEL/OPTION,6,TO,2
CIMFIL/OFF
$$
$$ CHECK FOR OUTPUT OF LIMIT AS FATAL ERROR
MW1=MACRO/
  DO/10,N1=1,501
    T1=TEXT/READ,CHECK,N1
    I1=INDXF(T1,(TEXT/'ERROR$EOF'))
    IF(I1 .GT. 0)JUMPTO/90
    I1=INDXF(T1,(TEXT/'X-AXIS POSITION OUTSIDE LIMITS'))
    IF(I1 .GT. 0)JUMPTO/20
    I1=INDXF(T1,(TEXT/'Y-AXIS POSITION OUTSIDE LIMITS'))
    IF(I1 .GT. 0)JUMPTO/20
  10)CONTIN
  20)CONTIN
  DMY=FILEF(0,1,(TEXT/'***FATAL ERROR*** TAPE FILE PURGED'))
  REWIND/PUNCH
  $$REWIND/CONVI,827     $$ IF PPWORD/PUNCH,n WAS USED
  INSERT/(TEXT/'***FATAL ERROR*** TAPE FILE PURGED$')
  90)CONTIN
TERMAC
$$
CIMFIL/ON,FINI
  DMY=POSTF(20)
  CALL/MW1
  DMY=POSTF(21)
  DMY=POSTF(13)
CIMFIL/OFF
```

6.22 FIL Example 22: How to support DIMS-CMM data from a PTC NCL file

The **DMIS** data is passed onto a file without interpreting the contents. It is up to the user to write FIL (see the **FIL Example** below) to parse this data as needed.

For each **DMIS/ON-OFF** pair in the NCL file, a new file named input_file001.dms,input_file002.dms etc will be created. The name **input_file** is the actual name of the input NCL file. The contents of the ***.dms** will be exact **DMIS** data from the NCL file.

G-Post then will see the modified **DMIS** data and will not fail with translation errors while reading **DMIS** data since they are in an external file.

The modified commands will be:

MDWRIT/ON
PPRINT DMSFILE=INPUT_FILE_NAMEnnn.DMS (nnn=DMIS/sequence number)
MDWRIT/OFF

Sample DMIS test file t01.ncl:

PARTNO / PPROC_DEV_01
UNITS / INCHES
LOADTL / 1
SPINDL / RPM, 4000.000000, CLW, MAXRPM, 4000.000000
FEDRAT / 40.000000, IPM
GOTO / 2.6313, -3.7250, 1.0000
DMIS/ON

PTMEAS / CART, 5.0394, -0.7500, -0.3750, $
1.0000000000, 0.0000000000, 0.0000000000
RAPID
GOTO / 5.2394, -0.7500, -0.3750
DMIS/OFF
GOTO / 2.6313, -3.7250, 1.0000
FINI

G-Post will create t01001.dms file:

PTMEAS / CART, 5.0394, -0.7500, -0.3750, $
1.0000000000, 0.0000000000, 0.0000000000
RAPID
GOTO / 5.2394, -0.7500, -0.3750

G-Post will see modified NCL data as below:

```
PARTNO / PPROC_DEV_01
UNITS / INCHES
LOADTL / 1
SPINDL / RPM, 4000.000000,  CLW,  MAXRPM, 4000.000000
FEDRAT / 40.000000,  IPM
GOTO / 2.6313, -3.7250, 1.0000
MDWRIT/ON
PPRINT DMSFILE=T01001.DMS
MDWRIR/OFF
GOTO / 2.6313, -3.7250, 1.0000
FINI
```

FIL Example:

```
$$ PTC NCL FILE WILL CONTAIN CMM-DMIS INFO AS BELOW
$$ DMIS/ON
$$ --DATA
$$ DMIS/OFF
$$
$$ GPOST WILL CONVERT THIS INTO
$$
$$ WE CAN READ THE DMS FILE AND WRITE TO TAPE FILE OR PARSE ETC
$$
CIMFIL/ON,MDWRIT
  IW4=POSTF(7,4)
  CASE/IW4
  WHEN/(ICODEF(ON))
    DMY=POSTF(14)       $$ GET DMS FILE NAME
    DMY=POSTF(13)
    TM1=TEXT/CLW
    TM1=TEXT/RANGE,TM1,10,66
    TM1=TEXT/LOW,TM1
    DMY=FILEF(1,2,TM1)  $$ OPEN DMS FILE
    PPRINT/'BEGIN DMS DATA'
    $$ DMS FILE READ LOOP
      LBM10)CONTIN
      TR1=TEXT/READ,1
      I1=CMPRF(TR1,(TEXT/'ERROR$EOF'))
      IF(I1 .EQ. 1)JUMPTO/LBM90
      $$ ***********************************************
      $$ ALTER TEXT STRING TR1 TO OUTPUT DESIRED DMS DATA
      $$ ***********************************************
      DMY=FILEF(4,1,TR1)  $$ WRITE TO TAP FILE
      JUMPTO/LBM10        $$ GET NEXT DMS FILE LINE
    $$ END OF DMS FILE
    LBM90)CONTIN
    DMY=FILEF(1,5,TM1)  $$ CLOSE DMS FILE
  WHEN/OTHERS
    PPRINT/'END DMS DATA'
  ENDCAS
CIMFIL/OFF
```

7 Vocabulary Codes

Introduction

This chapter contains the Austin N.C., Inc. vocabulary words and their associated codes. The first section is in numerical order, while the second is in alphabetical order.

7.1 Numerical Order

1.....ATANGL
2.....CENTER
3.....CROSS
4.....FUNOFY
5.....INTOF
6.....INVERS
7.....LARGE
8.....LEFT
9.....LENGTH
10.....MINUS
11.....NEGX
12.....NEGY
13.....NEGZ
14.....NOX
15.....NOY
16.....NOZ
17.....PARLEL
18.....PERPTO
19.....PLUS
20.....POSX
21.....POSY
22.....POSZ
23.....RADIUS
24.....RIGHT
25.....SCALE
26.....SMALL
27.....TANTO
27.....DSTAN
28.....TIMES
29.....TRANSL
30.....UNIT
31.....XLARGE
32.....XSMALL
33.....XYPLAN
34.....XYROT
35.....YLARGE
36.....YSMALL
37.....YZPLAN
38.....YZROT
39.....ZLARGE
40.....ZSMALL
41.....ZXPLAN
42.....ZXROT
43.....3PT2SL
44.....4PT1SL
45.....5PT
46.....INTERC
47.....SLOPE
48.....IN
49.....OUT
51.....ALL
52.....LAST
53.....NOMORE
54.....SAME
55.....MODIFY
56.....MIRROR
57.....START
58.....ENDARC
59.....CCLW
60.....CLW
61.....MEDIUM
62.....HIGH
63.....LOW
64.....CONST
65.....DECR
66.....INCR
67.....GRID
68.....ROTREF
69.....TO
70.....PAST
71.....ON
72.....OFF
73.....IPM
74.....IPR
75.....CIRCUL
76.....LINEAR
77.....PARAB
78.....RPM
79.....MAXRPM
80.....TURN
81.....FACE
82.....BORE
83.....BOTH
84.....XAXIS
85.....YAXIS
86.....ZAXIS

87..... [Unassigned]
88.....AUTO
89.....FLOOD
90.....MIST
91.....TAPKUL
92.....STEP
93.....MAIN
93.....RAIL
94.....SIDE
95.....LINCIR
96.....MAXIPM
97.....REV
98.....TYPE
99.....NIXIE
100.....LIGHT
101.....FOURPT
102.....TWOPT
103.....PTSLOP
104.....PTNORM
105.....SPLINE
106.....RTHETA
107.....THETAR
108.....XYZ
109.....MWD109
110.....TRFORM
111.....NORMAL
112.....UP
113.....DOWN
114.....LOCK
115.....SFM
116.....XCOORD
117.....YCOORD
118.....ZCOORD
119.....MULTRD
120.....XYVIEW
121.....YZVIEW
122.....ZXVIEW
123.....SOLID
124.....DASH
125.....DOTTED
126.....CTLIN
127.....DITTO
128.....PEN
129.....SCRIBE
130.....BLACK
131.....RED
132.....GREEN
133.....BLUE
134.....INTENS
135.....LITE
136.....MED
137.....DARK
138.....CHUCK
139.....COLLET
140.....AAXIS
141.....BAXIS
142.....CAXIS
143.....TPI
144.....OPTION
145.....RANGE
146.....PSTAN
147.....MWD
148.....FRONT
149.....REAR
150.....SADDLE
151.....MILL
152.....THRU
153.....DEEP
154.....TRAV
155.....SETOOL
156.....SETANG
157.....HOLDER
158.....MANUAL
159.....ADJUST
160.....CUTANG
161.....NOW
162.....NEXT
163.....DRILL
164.....BINARY
165.....BCD
166.....NEUTRL
167.....TLPOT
168.....TAP
169.....OXYGEN
170.....OPTAB
171.....PREHET
172.....TORCH
173.....MWD173
175.....MLSTOP
176.....SETREF
177.....TABLE
178.....MAGZIN
179.....TURET
180.....MASTER
181.....SLAVE
182.....ARC
183.....RANDOM
184.....RETAIN
185.....ZIGZAG
186.....OMIT
187.....AVOID
188.....DELTA
189.....AT
190.....BOREDS
191.....BORED
192.....PULBAC
193.....FINE
194.....NOBACK
195.....COARSE
196.....SIZE
197.....DWELLP
198.....MWD198

199.....MWD199
200.....MILLRP
201.....MILLFD
202.....MWD202
203.....MWD203
204.....ZFEED
205.....DIAMTR
206.....ZRAPID
207.....BAR
208.....DEEPCL
209.....BORE5
210.....BORE6
211.....BORE7
212.....BORE8
213.....BORE9
214.....BORE10
215.....AVOIDP
216.....DEEPKB
217.....FEEDX
218.....FEEDY
219.....FEEDZ
220.....MILLOC
221.....MWD221
222.....MWD222
223.....MWD223
224.....MWD224
225.....MWD225
226.....MWD226
227.....MWD227
228.....MWD228
229.....MWD229
230.....MWD230
231.....OPER
232.....DRAW
233.....INTGER
234.....HOLRTH
235.....SYMBOL
236.....NOSFM
237.....CAM
238.....HED
239.....PALLET
240.....TUL
241.....READER
242.....INDEXR
243.....BORESS
244.....SPDRL
245.....CBORE
246.....ORIENT
247.....TILT
248.....ROCK
249.....SHIFT
250.....SCHEDL
250.....TIPDIA
251.....COBORE
252.....MWD252
253.....MWD253
254.....MWD254
255.....MWD255
256.....CSINK
257.....MWD257
258.....MWD258
259.....MWD259
260.....PART
261.....MWD261
262.....REAM
263.....MWD263
264.....MWD264
265.....MWD265
266.....MWD266
267.....MWD278
268.....DRLTYP
269.....TULTYP
270.....FEED
271.....NOEDGE
272.....SPEED
273.....TLLIFE
274.....TLMAT
275.....OFSETL
276.....CORMIL
277.....MWD277
278.....DRAG
279.....DWELL
280.....RAPTO
281.....FEDTO
282.....KEYLCK
283.....MWD283
284.....MWD284
285.....MWD285
286.....MWD286
287.....QUILL
288.....MWD288
289.....MWD289
290.....MWD290
291.....MWD291
292.....CONVI
293.....CONVF
294.....CONVE
295.....MWD295
296.....MILMET
297.....CENMET
298.....TPF
301.....MM
302.....CM
303.....INCHES
304.....FEET
315.....MMPM
316.....MMPR
401.....TAPER
402.....FILLET
403.....CORNER
404.....ENDTRN
405.....ENDFAC

406.....QUAD1
407.....QUAD2
408.....QUAD3
409.....QUAD4
410.....REPOS
411.....ELSE
412.....MOVETO
505.....SMM
601.....POINT
602.....LINE
603.....PLANE
604.....CIRCLE
605.....CYLNDR
606.....ELLIPS
607.....HYPERB
608.....CONE
609.....GCONIC
610.....LCONIC
611.....VECTOR
612.....MATRIX
613.....SPHERE
614.....QADRIC
615.....POLCON
616.....LOFT
617.....TOOL
618.....PATERN
619.....DATA
620.....TEXT
650.....TABCYL
653.....RLDSRF
701.....FROM
702.....GODLTA
703.....GOTO
704.....GO
705.....OFFSET
706.....INDIRP
707.....INDIRV
708.....SRFVCT
709.....GOLFT
710.....GORGT
711.....GOFWD
712.....GOBACK
713.....GOUP
714.....GODOWN
715.....TLLFT
716.....TLRGT
717.....TLON
718.....TLONPS
719.....TLOFPS
720.....PSIS
721.....TOLER
722.....INTOL
723.....OUTTOL
724.....2DCALC
725.....3DCALC
726.....NDTEST
727.....CUT
728.....DNTCUT
729.....UTON
730.....UTOFF
731.....TLAXIS
732.....MULTAX
733.....MAXDP
734.....NUMPTS
735.....THICK
736.....NOPS
737.....AUTOPS
738.....GOUGCK
739.....UNITS
740.....NORMPS
741.....NORMDS
742.....CUTTER
801.....TRANTO
802.....POCKET
803.....REFSYS
804.....RESERV
805.....LOOPST
806.....DEBUG
807.....TUNEUP
808.....ZSURF
809.....LOOPND
810.....TERMAC
811.....DO
812.....NOPOST
813.....CLPRNT
814.....PTONLY
815.....CONTRL
816.....NOPLOT
817.....TIMLIM
818.....LATHSQ
819.....THRDSQ
820.....SETAX
821.....SBOUND
822.....EBOUND
823.....IF
824.....JUMPTO
825.....CONTIN
826.....PRINT
827.....PUNCH
828.....READ
829.....XREF
830.....UTURN
901.....SQRTF
902.....SINF
903.....COSF
904.....EXPF
905.....LOGF
906.....ATANF
907.....ABSF
908.....TANF
909.....INTF
910.....FIXF

911.....FLOATF
921.....LNTHF
922.....DOTF
923.....NUMF
924.....ANGLF
925.....DISTF
926.....ATAN2F
927.....SIGNF
928.....MODF
929.....CANF
930.....MAXF
931.....MINF
1001.....PLUNGE
1002.....HEAD
1003.....MODE
1004.....CLEARP
1005.....TMARK
1006.....REWIND
1007.....CUTCOM
1008.....REVERS
1009.....FEDRAT
1010.....DELAY
1011.....AIR
1012.....OPSKIP
1012.....DELETE
1013.....LEADER
1014.....PPLOT
1015.....MACHIN
1016.....MCHTOL
1017.....PIVOTZ
1018.....MCHFIN
1019.....SEQNO
1020.....INTCOD
1021.....DISPLY
1022.....AUXFUN
1023.....CHECK
1024.....POSTN
1025.....TOOLNO
1026.....ROTABL
1027.....ORIGIN
1028.....SAFETY
1029.....ARCSLP
1030.....COOLNT
1031.....SPINDL
1032.....DEBUGG
1033.....TURRET
1034.....POSMAP
1035.....ROTHED
1036.....THREAD
1037.....TRANS
1038.....TRACUT
1039.....INDEX
1040.....COPY
1041.....PLOT
1042.....OVPLOT
1043.....LETTER
1044.....PPRINT
1045.....PARTNO
1046.....INSERT
1047.....CAMERA
1048.....PREFUN
1049.....COUPLE
1050.....PITCH
1051.....MDWRIT
1052.....MDEND
1053.....ASLOPE
1054.....CYCLE
1055.....LOADTL
1056.....SELCTL
1057.....CLRSRF
1058.....MJ1058
1059.....DRAFT
1060.....CLAMP
1061.....PLABLE
1062.....MAXDPM
1063.....SLOWDN
1064.....MAXVEL
1065.....LPRINT
1066.....ROTATE
1067.....LINTOL
1068.....PPTOL
1069.....PRFSEQ
1070.....VTLAXS
1071.....COMBIN
1072.....POSITN
1073.....OP
1074.....SELECT
1075.....LOAD
1076.....MJ1076
1077.....DOHOLE
1078.....LIMIT
1079.....MATERL
1080.....TLSPEC
1081.....MJ1081
1082.....PUNCHP
1083.....REPEAT
1084.....XOFSET
1085.....TLUSE
1086.....OBRCNT
1087.....SET
1088.....MJ1088
1089.....MJ1089
1090.....MJ1090
1091.....MJ1091
1092.....MJ1092
1093.....MJ1093
1094.....MJ1094
1095.....TITLES
1096.....MJ1096
1097.....WCORN
1098.....WAREA
1099.....WLNGTH

1201.....END
1202.....STOP
1203.....OPSTOP
1204.....ISTOP
1205.....RAPID
1206.....SWITCH
1206.....EXIT
1207.....RETRCT
1208.....DRESS
1209.....PICKUP
1210.....UPLOAD
1211.....PENUP
1212.....PENDWN
1213..... [Unassigned]
1214.....CODEL
1215.....RESET
1216.....BREAK
1217.....GOHOME
1301.....SYN
1302.....CALL
1303.....MACRO
1304.....OBTAIN
1305.....CANON
1305.....REDEF
1306.....FINI
1307.....CLDATA
1308.....INCLUD
1309.....PPWORD

7.2 Alphabetical Order

8 REPLAC Command

REPLAC is a FIL **ONLY** command that allows you to edit/modify the tape output block, similar to a text editor during the post processor execution.

8.1 REPLAC/t1, t2[,n1, n2]

t1 = the target string to be modified
t2 = the replacement string
n1 = number of time to replace the string in the current tape block.
n2 = number of tape blocks to replace.
If n1=0 or is not given, all occurrences in the tape block are replaced.
If n2=0 or is not given, all occurrences in all tape blocks are replaced.

Example:

```
$$ CHANGE A360.0 TO A0.0
T1=TEXT/'A360.0'
T2=TEXT/'A0.0'
REPLAC/T1,T2
```

8.2 REPLAC/t1, t2, t3[,n1, n2]

t1 = the target string to be modified
t2 = the replacement string
t3 = wild card to be used.
n1 = will be set to 1, for wild card option.
n2 = number of tape blocks to replace.

Example:

```
$$ CHANGE + OR - 360 TO 0
T1=TEXT/'A*360.0'
T2=TEXT/'A0.0'
T3=TEXT/'*'
REPLAC/T1,T2,T3,1
```

8.3 REPLAC: Special Wild Card Option

If t1 is blank then t2 is added to the beginning of the tape block.
If t1 is the wild card then t2 is added to the end of the tape block.
If t2 is '**TO_PASTE_BUF'** the t1 string found in the tape block will be written to the paste buffer.

If t2 is **'FROM_PASTE_BUF'** the string in the paste buffer will replace the t1 string in the tape block.

Example:

```
$$ Switch XY TO YZ
REPLAC/(TEXT/'X*'),(TEXT/'YTO_PASTE_BUF'),(TEXT/'*')
REPLAC/(TEXT/'Y*'),(TEXT/'Y*XFROM_PASTE_BUF'),(TEXT/'*')
```

If t1 is 'any text*' that ends with the wild card, then the number following that text will be the target string.

Example:

```
$$ ADD C0B0 TO A
T1=TEXT/'A*'
T2=TEXT/'C0B0A*'
T3=TEXT/'*'
REPLAC/T1,T2,T3
```

8.4 REPLAC/*t1*,*t2*,PLUS-MINUS,[ON-OFF]

t1 = the target string in tape block
t2 = the new string to be inserted
PLUS = insert t2 after current tape block
MINUS = insert t2 before current tape block
ON-OFF= [optional] to control the sequence number output for the new blocks.

Note: You must have SEQNO in effect - if not ON-OFF will be ignored.
Default is ON to output sequence numbers.

Multiple blocks can be inserted by repeating the **REPLAC/*t1*** commands.

Example:

1. For each M06 block, precede it with a G28 block and follow it with a G41 block:

```
REPLAC/(TEXT/'M06'),(TEXT/'G28 X0 Y0 Z10'),MINUS
REPLAC/(TEXT/'M06'),(TEXT/'G41 D1'),PLUS
```

2. Before M30, output T00 without a sequence number:

```
REPLAC/(TEXT/'M30'),(TEXT/'T00'),MINUS,OFF
```

8.5 REPLAC/OFF

Turn off all **REPLAC** commands and clear the **REPLAC** table.

8.6 Special Notes on the REPLAC Command.

a) **REPLAC** is a **FIL ONLY** command.
   All text strings must be defined as **TEXT/** or nested **TEXT** definitions.
b) **REPLAC** commands can be inside a **CIMFIL/ON CIMFIL/OFF** routine or in the **GLOBAL** area of the **FIL** file.
c) For wild card **REPLAC**, the wild card string cannot be in the tape block.
d) **REPLAC** is executed as the last action prior to writing the tape block to the tape file.
e) Replaced tape blocks cannot be greater than 128 characters.
f) **REPLAC** only modifies the tape block and not the LST file.
g) A maximum of 1000 **REPLAC** commands may be active at one time.
h) **REPLAC/OFF** will clear the **REPLAC** table and 1000 more **REPLAC** commands will be available.
i) If **PRINT/ON,REPLAC** is in effect the **REPLAC** table will be printed for debugging in the LST file.
   The default is not to print any **REPLAC** debugging information into the LST file.

9 _MCDWT Macro

9.1 Definition:

The **_MCDWT** (MCD Write) macro is used to edit the current output block just before it is written to the output files. There are two output files written to by the G-POST, the LST file and the MCD file.

To enable the **_MCDWT** macro, **INTCOM(1920)** must be set to 2. This can be set in the Option File Generator on the "General" tab of the "Start/End of Program" panel or it can be set directly in the FIL file by using the command: **RSLT = POSTF(2,1,1920,2)** to enable and **RSLT = POSTF(2,1,1920,0)** to disable. Once set, the **_MCDWT** macro **must** be present somewhere in the FIL file, and will be called every time the G-POST is ready to output a block to the output files.

9.2 Implementation:

These two **TEXT** commands, **T1=TEXT/LAST,3 (.LST Block)** and **T2=TEXT/LAST,*4* (MCD Block)** are used to get the contents of the output block. See the **TEXT/LAST** section of this manual for more details.

***t1* = TEXT/LAST,3**

t1: **Text string for the current output to the LIST file**

The **LAST,3** option reads the current output buffer for the LST file. This is the string that would have been written to the LIST file if the _MCDWT macro had not been called by FIL.

***t1* = TEXT/LAST,4**

t1: **Text string for the current output to the MCD file**

The **LAST,4** option reads the current output buffer for the MCD file. This is the string that would have been written to the MCD file if the _MCDWT macro had not been called by FIL.

The **_MCDWT** macro **must** be defined within the current FIL file.

9.3 _MCDWT Examples:

9.3.1 *Sample Macro: (see the file _MCDWT.FIL, supplied with the system)*

```
$$ 09-02-00 AUSTIN N.C., INC.; CREATE _MCDWT MACRO
$$
$$ G-POST WILL CALL THIS MACRO JUST BEFORE WRITING THE MCD BLOCK
$$ TO THE PUNCH/TAPE FILE - WHEN PLABEL/OPTION,90,TO,2 IS SET.
$$ T1=TEXT/LAST,4 COMMAND WILL RETURN THE CURRENT BLOCK IN T1.
$$
$$ USING THIS MACRO YOU CAN:
$$ 1. EDIT THE CURRENT BLOCK WITH ANY TEXT/CMD
$$ 2. OUTPUT ADDITIONAL BLOCKS WITH FILEF(4,1,TEXT)
$$ 3. SKIP THE CURRENT BLOCK
$$
$$ YOU CAN USE ANY FIL COMMAND, POSTF, FILEF ETC. INSIDE THIS MACRO
$$ ALL POST COMMANDS ARE IGNORED INCLUDING INSERT OR POSTF(13).
$$
$$ THIS SAMPLE MACRO ADDS PGM TO START AND END OF TAPE AND
$$ A MSG FOR TOOL CHANGE. OTHERWISE WRITES THE BLOCK AS IS.
REDEF/ON
_MCDWT=MACRO/
  T4=TEXT/LAST,4                                $$ GET CURRENT BLOCK
  I1=INDXF(T4,(TEXT/'%'))                       $$ ADD PGM START FOR 1ST %
  IF(I1 .EQ. 1) THEN
      I1=POSTF(7,2)                             $$ CHECK IF IT IS FINI
      IF(I1 .EQ. 14000) THEN
          DMY=FILEF(4,1,T4)
          DMY=FILEF(4,1,(TEXT/'(PGM=END)'))
      ELSE                                      $$ START OF TAPE
          DMY=FILEF(4,1,(TEXT/'(PGM=START)'))
          DMY=FILEF(4,1,T4)
      ENDIF
      JUMPTO/LB90
  ENDIF
  I1=INDXF(T4,(TEXT/'M06'))                     $$ IF TOOL CHANGE ADD MSG
  IF(I1 .GT. 0) THEN
      DMY=FILEF(4,1,(TEXT/'(MSG=TOOL CHANGE)'))
      T4=TEXT/MODIFY,T4,(TEXT/'M06'),(TEXT/'M66'),1
      DMY=FILEF(4,1,T4)
      JUMPTO/LB90
  ENDIF
  DMY=FILEF(4,1,T4)                             $$ ANY BLOCK OUTPUT AS IS
  LB90)CONTIN
TERMAC
```

9.3.2 Sample Input/Output*:*

Input file:

```
PARTNO TEST
MACHIN/UNCX01,1
LOADTL/1
SPINDL/300
COOLNT/ON
FEDRAT/10
FROM/10,10,10
GOTO/1,1,1
GOTO/10,10,10
LOADTL/2
SPINDL/300
COOLNT/ON
GOTO/2,2,2
END
FINI
```

MCD file:

```
(PGM=START)
%
(MSG=TOOL CHANGE)
N1T01M66
N2M41
N3S00300M03
N4M07
N5G1X1.Y1.Z1.F10.
N6X10.Y10.Z10.
(MSG=TOOL CHANGE)
N7T02M66
N8S00300M03
N9M07
N10G1X2.Y2.Z2.F10.
N11M02
%
(PGM=END)
```

10 _OUTPT Macro

10.1 Definition:

The **_OUTPT** (Output Editor) macro is used to edit the contents of the WORD buffer **DBLCOM(513-564)** for Mill and **DBLCOM(1374-1425)** for Lathe. As the G-Post is processing CL commands, it loads the WORD buffer with the desired output values for each letter address. When the "Output" function in G-Post is executed, it reads the contents of the WORD buffer, formats the data appropriately for the desired address and writes the output block to the MCD and listing (LST) files. By manipulating the contents of the WORD buffer prior to the G-Post executing the "Output" function, you can precisely control the desired output. After the _OUPPT macro has completed it is possible for the G-Post to execute the _MCDWT macro to further modify data to the MCD file.

The WORD buffer stores the desired output values for each letter address and verify letter address as follows:

Letter Addresses (Mill Post)

A = DBLCOM(513)	B = DBLCOM(514)	C = DBLCOM(515)	D = DLBCOM(516)
E = DBLCOM(517)	F = DBLCOM(518)	G = DBLCOM(519)	H = DLBCOM(520)
I = DBLCOM(521)	J = DBLCOM(522)	K = DBLCOM(523)	L = DLBCOM(524)
M = DBLCOM(525)	N = DBLCOM(526)	O = DBLCOM(527)	P = DLBCOM(528)
Q = DBLCOM(529)	R = DBLCOM(530)	S = DBLCOM(531)	T = DLBCOM(532)
U = DBLCOM(533)	V = DBLCOM(534)	W = DBLCOM(535)	X = DLBCOM(536)
Y = DBLCOM(537)	Z = DBLCOM(538)		

Verify Letter Addresses (Mill Post) (verify letter addresses are used in the .LST file in columns 80-132)

A = DBLCOM(539)	B = DBLCOM(540)	C = DBLCOM(541)	D = DLBCOM(542)
E = DBLCOM(543)	F = DBLCOM(544)	G = DBLCOM(545)	H = DLBCOM(546)
I = DBLCOM(547)	J = DBLCOM(548)	K = DBLCOM(549)	L = DLBCOM(550)
M = DBLCOM(551)	N = DBLCOM(552)	O = DBLCOM(553)	P = DLBCOM(554)
Q = DBLCOM(555)	R = DBLCOM(556)	S = DBLCOM(557)	T = DLBCOM(558)
U = DBLCOM(559)	V = DBLCOM(560)	W = DBLCOM(561)	X = DLBCOM(562)
Y = DBLCOM(563)	Z = DBLCOM(564)		

Letter Addresses (Lathe Post)

A = DBLCOM(1374)	B = DBLCOM(1375)	C = DBLCOM(1376)	D = DLBCOM(1377)
E = DBLCOM(1378)	F = DBLCOM(1379)	G = DBLCOM(1380)	H = DLBCOM(1381)
I = DBLCOM(1382)	J = DBLCOM(1383)	K = DBLCOM(1384)	L = DLBCOM(1385)
M = DBLCOM(1386)	N = DBLCOM(1387)	O = DBLCOM(1388)	P = DLBCOM(1389)
Q = DBLCOM(1390)	R = DBLCOM(1391)	S = DBLCOM(1392)	T = DLBCOM(1393)
U = DBLCOM(1394)	V = DBLCOM(1395)	W = DBLCOM(1396)	X = DLBCOM(1397)
Y = DBLCOM(1398)	Z = DBLCOM(1399)		

Verify Letter Addresses (Lathe Post) (verify letter addresses are used in the .LST file in columns 80-132)

A = DBLCOM(1400)	B = DBLCOM(1401)	C = DBLCOM(1402)	D = DLBCOM(1403)
E = DBLCOM(1404)	F = DBLCOM(1405)	G = DBLCOM(1406)	H = DLBCOM(1407)
I = DBLCOM(1408)	J = DBLCOM(1409)	K = DBLCOM(1410)	L = DLBCOM(1411)
M = DBLCOM(1412)	N = DBLCOM(1413)	O = DBLCOM(1414)	P = DLBCOM(1415)
Q = DBLCOM(1416)	R = DBLCOM(1417)	S = DBLCOM(1418)	T = DLBCOM(1419)
U = DBLCOM(1420)	V = DBLCOM(1421)	W = DBLCOM(1422)	X = DLBCOM(1423)
Y = DBLCOM(1424)	Z = DBLCOM(1425)		

Note: The above noted **DBLCOM** locations are given for your reference only. **POSTF(31,1-2,--)** functions will get/set the WORD buffer using A-Z as the index and not the actual **DBLCOM** location number.

To enable the **_OUTPT** macro, **INCTOM(4667)** must be set to 1. This can be set directly in the Option File Generator on the "General" tab of the "Start/End of Program" panel or in the FIL file by using the command: **RSLT = POSTF(2,1,4667,1)** to enable and **RSLT = POSTF(2,1,4667,0)** to disable.

10.2 Implementation:

Once set, the **_OUTPT** macro **must** be present somewhere in the FIL file, and will be called prior to the G-Post executing it's "Output" function every time.

The _OUTPT macro is different than the _MCDWT as you are working with output block scalars instead of text and can avoid string parsing.

Using _OUTPT you can:

1. Edit the current output block using the POSTF(31,--) functions
2. Output additional blocks with POSTF(31,19) or FILEF(4,1,text)
3. Skip the current block

Notes:

1. Only a command that triggers the G-Post "Output" function can be edited, such as GOTO, POSTN, LOADTL etc. RAPID is a post command but does not trigger the "Output" function, so _OUTPT would not be executed.
2. String output commands like INSERT and PPRINT do not trigger the G-Post "Output" function and the _OUTPT would not be executed.
3. The POSTF(31,--) functions are only allowed to be used inside the _OUTPT macro.
4. The REPLAC/cmd commands are applied to the output string after the _OUTPT has been completed.
5. You can used any FIL command within the _OUTPT macro, such as POSTF() and FILEF().
6. All post commands inside the _OUTPT macro are ignored, including INSERT and POSTF(13).
7. The _MCDWT macro is applied after the _OUTPT macro has completed.

POSTF(31)

The POSTF(31) functions are designed to work within the _OUTPT macro and allow you to manipulate the WORD buffer contents and output. Attempting to use these functions outside of the _OUTPT macro will generate a FIL error.

RSLT = POSTF(31,1,arg2)

This POSTF function is used to GET (retrieve) a value from the WORD buffer for the desired letter address. ***arg2*** specifies the desired address 1-26 for A-Z and 27-52 for Verify A–Verify Z. ***RSLT*** will contain the current value stored in the WORD buffer for the desired ***arg2*** address. If the desired address is empty then ***RSLT*** will be set to 999999.

RSLT = POSTF(31,2,arg2,arg3)

This POSTF function is used to SET (load) a value in the WORD buffer for the desired letter address. ***arg2*** specifies the desired address 1-26 for A-Z and 27-52 for Verify A–Verify Z. ***arg3*** specified the desired value to be loaded into the WORD buffer for the desire ***arg2*** address. ***RSLT*** will be set to Zero.

RSLT = POSTF(31,3)

This POSTF function is used to clear the entire WORD buffer. All WORD buffer locations will be set to empty (99999.). ***RSLT*** will be set to Zero.

RSLT = POSTF(31,19)

This POSTF function is used execute the "Output" function of the G-Post. It will process the current WORD buffer. ***RSLT*** will be set to Zero.

RSLT = POSTF(31,20)

This POSTF function is used save a temporary copy of the WORD buffer. ***RSLT*** will be set to Zero.

RSLT = POSTF(31,21)

This POSTF function is used reload the WORD buffer with the contents of the last saved copy of the WORD buffer from a previous POSTF(31,20) command. ***RSLT*** will be set to Zero.

10.3 _OUTPT Examples:

10.3.1 *Sample Macro: (see the file _OUTPT.FIL, supplied with the system)*

```
$$ 06-15-04 AUSTIN N.C., INC.-AUSTIN; CREATE _OUTPT SAMPLE MACRO
$$
$$ G-POST WILL CALL THIS MACRO JUST BEFORE PROCESSING AN OUTPUT
$$ BLOCK WHEN I4667=1. THIS EDIT MACRO IS DIFFERENT FROM _MCDWT
$$ AS YOU ARE WORKING WITH OUTPUT BLOCK SCALARS INSTEAD OF TEXT
$$ AND CAN AVOID STRING PARSING.
$$
$$ USING THIS MACRO YOU CAN:
$$
$$ 1. TO EDIT THE CURRENT BLOCK WITH ANY POSTF()
$$ 2. OUTPUT ADDITIONAL BLOCKS WITH POSTF(31,19) OR FILEF(4,1,TEXT)
$$ 3. SKIP THE CURRENT BLOCK
$$
$$ NOTE:
$$
$$ 1. ONLY A POST OR GOTO COMMAND THAT TRIGGERED OUTPUT
$$   INCLUDING POSTN CAN BE EDITED
$$ 2. STRING OUTPUT LIKE INSERT,PPRINT ARE BY PASSED
$$ 3. POSTF(31) IS ALLOWED ONLY WITHIN _OUTPT MACRO
$$ 4. REPLAC/CMD IS APPLIED AFTER _OUTPT MACRO
```

```
$$ 5. YOU CAN USE ANY FIL COMMAND,POSTF,FILEF INSIDE THIS MACRO
$$ 6. ALL POST COMMANDS INSIDE THE MACRO ARE IGNORED INCLUDING
$$    INSERT OR POSTF(13).
$$
$$ THIS SAMPLE MACRO HAS 4-EXAMPES TO ALTER/ADD/SKIP BLOCKS
$$
$$ EXAMPLE-1 FOR TOOLCHG M06 OUTPUT N9999 G28 X0 Y0 Z0 BEFORE
$$ EXAMPLE-2 CHANGE C-270 TO C90
$$ EXAMPLE-3 FOR G43 ADD Z-CURRENT VALUE IF Z IS NOT THERE
$$ EXAMPLE-4 SKIP G93 BLOCK FROM OUTPUT
$$
REDEF/ON
$$
EMT=999999                          $$ DEFINE EMPTY
EPS=0.00001                         $$ SMALL TOL FOR IF TEST
CONTRL/TOLER,IF,EPS                 $$ SET IF/EQ TEST TOLERANCE
_OUTPT=MACRO/
  MWRD=POSTF(31,1,13)               $$ 1)CHK M06 GET M-WORD
  IF(MWRD .EQ. 6) THEN
  DMY=POSTF(31,20)                  $$ SAVE M06 BLOCK
    DMY=POSTF(31,03)                $$ CLEAR BLOCK
    DMY=POSTF(31,2,14,9999)         $$ SET N9999
    DMY=POSTF(31,2,7,28)            $$ G28 X0 Y0 Z0
    DMY=POSTF(31,2,24,0)
    DMY=POSTF(31,2,25,0)
    DMY=POSTF(31,2,26,0)
    DMY=POSTF(31,19)                $$ OUTPUT G28 BLOCK
    DMY=POSTF(31,21)                $$ RESTORE M06 BLOCK
    JUMPTO/LB80
  ENDIF
  CWRD=POSTF(31,1,3)                $$ 2)CHK C-270 GET C-WORD
  IF(CWRD .EQ. (-270)) THEN         $$ CHG C-270 TO C90
    DMY=POSTF(31,2,3,90)
  ENDIF
  GWRD=POSTF(31,1,7)                $$ 3)CHK G43 GET G-WORD
  IF(GWRD .EQ. 43) THEN
    ZWRD=POSTF(31,1,26)             $$ GET Z-WORD
    IF(ZWRD .EQ. EMT) THEN          $$ IF NO Z SET CURRENT Z
      R=POSTF(1,3,(291+26))
      =POSTF(31,2,26,ZCUR)
    ENDIF
    JUMPTO/LB80
  ENDIF
  IF(GWRD .EQ. 93) THEN             $$ 4)SKIP A G93 BLOCK
    JUMPTO/LB90
  ENDIF
  LB80)CONTIN
  DMY=POSTF(31,19)                  $$ OUTPUT CURRENT BLOCK
  LB90)CONTIN                       $$ BRANCH LB90 TO SKIP OUTPUT
TERMAC
```

10.3.2 Sample Input/Output*:*

Input file:

```
PARTNO TEST
MACHIN/UNCX01,1
LOADTL/22,ADJUST,22
SPINDL/400
COOLNT/ON
FROM/10,10,10
RAPID
GOTO/1,1,1
FEDRAT/10
GOTO/10,10,10
LOADTL/44,ADJUST,44
SPINDL/500
COOLNT/ON
RAPID
GOTO/2,2,2
FEDRAT/20
GOTO/20,20,20
END
FINI
```

MCD file:

```
%
M9999G28X0.Y0.Z0.
N1T22M6
N2M41
N3S400,M3
N4M7
N5G0X1.Y1.
N6G43 Z1.H22
N7G1X10.Y10.Z10.
M9999G28X0.Y0.Z0.
N8T44M6
N9S500,M3
N10M7
N11G0X2.Y2.
N12G43Z2.H44
N13G1X20.Y20.Z20.F20
N14M2
%
```

11 _REPOS Macro

11.1 Definition:

The _REPOS Macro is used to generate rotary axis re-positioning moves from within FIL once a rotary axis limit has been violated.

With “Check Axis Limits” set, if an axis, linear or rotary, is analyzed to be “Out of Limits” the G-Post will generate a warning message but no addition correction will be made.

For 5 axis machines, the rotary axes usually have several correct resolves for the specified tool axis. Since the G-Post always uses the shortest move, the calculated resolve might violate an axis limit. Once the axis limit is violated, the _REPOS macro can be called or the G-Post can automatically attempt to correct the problem.

Using the Option File Generator, Go to the “Machine Tool Type” panel and select the “Axes” tab. On this tab you will have the choice of how you want the post to deal with axis limit checking and automatic repositioning . It is recommended; you use the automatic correction method unless you want explicit control using your own reposition logic.

If you choose “Use automatic repositioning”: You will have the choice having the G-Post automatically perform the reposition of the rotary axis or to call the _REPOS macro.

Calling the _REPOS macro will allow you define the rules of the repositioning moves.

Rapid and feed rate moves are handled separately but both will call the same _REPOS macro if set to do so. When writing the _REPOS macro you must determine whether this was a rapid or feed rate move and work accordingly.

The _REPOS macro definition **must** reside in the FIL file.

The **INTCOM(1849) [PLABEL/LABEL,19]** controls the limit test. When set to 0, no limit checking will be performed.

When set to 1, standard limit checking will be performed and the warnings will be output to the listing file.

When set to 2, limit checking will be performed and an auto re-position function will be executed (Automatic or _REPOS macro).

11.2 Implementation:

For each GOTO point CL record, the axis limit is tested. Since each point has to be tested prior to output, it will take longer to process.

If the A, B or C axis is out of limits then and the _REPOS macro is enabled a FIL macro named _REPOS will be called to re-position the axis. If the _REPOS macro is not defined, a warning is output.

Note for APT users:

The APT system can generate multi-point GOTO records in the CL file. The auto limit check applies only to single GOTO points records - which is the default of most CAD systems. Optionally, you can set **INTCOM(1849)** to 3. This will convert the CL file to contain only single point GOTO point CL records.

INTCOM(1849) [PLABEL/LABEL,19] can only be set in the Option file.

11.3 _REPOS Examples:

11.3.1 *Sample Macro: (see the file _REPOS.FIL, supplied with the system)*

```
$$ 04-15-99 AUSTIN N.C., INC.: CREATE_REPOS MACRO
$$
$$ G-POST WILL CALL THIS MACRO FOR ABC-AXIS LIMITS IN A 5-AXES CASE
$$ WHEN PLABEL/OPTION,19,TO,2 IS SET
_REPOS=MACRO/
DMY=POSTF(20)                       $$ SAVE THE CURRENT CL RECORD
RTDIS=1                             $$ RETRACT Z-DISTANCE

X1=POSTF(1,3,911)           $$ GET LAST RAW CLPOS - NOT USED
Y1=POSTF(1,3,912)
Z1=POSTF(1,3,913)
I1=POSTF(1,3,914)
J1=POSTF(1,3,915)
K1=POSTF(1,3,916)

A1=POSTF(1,3,292)           $$ GET ABC POSITION FROM PRES(A-C)
B1=POSTF(1,3,293)
C1=POSTF(1,3,294)

ICY=POSTF(1,1,482)          $$ GET CYCLE FLAG

IWN=POSTF(1,1,1932)         $$ UPDATE WARNING COUNT
IWN=IWN+1
DMY=POSTF(2,1,1932,IWN)

IPRM=POSTF(1,1,2081)-3              $$ GET PRIMARY AXIS
ILM=0                       $$ FIND AXIS THAT HIT LIMIT - NOT USED

IF(POSTF(1,1,1) .NE. 53) THEN
                IF(A1.LT. (POSTF(1,3,595)))ILM=1
                IF(A1.GT. (POSTF(1,3,601)))ILM=1
ENDIF
IF(POSTF(1,1,2) .NE. 53) THEN
        IF(B1.LT. (POSTF(1,3,596)))ILM=2
        IF(B1.GT. (POSTF(1,3,602)))ILM=2
ENDIF
```

```
IF(POSITF(1,1,3) .NE.53) THEN
        IF(CL.LT. (POSTF(1,3,597)))ILM=3
        IF(C1.GT. (POSTF(1,3,603)))ILM=3
ENDIF
IF(ICY.EQ.1) THEN                       $$ KILL CYCLE
                CYCLE/OFF
ENDIF
DISPLY/NEXT
PPRINT/'***WARNING*** AXIS RE-POSITION BY TRAVEL LIMIT'
GODLTA/RTDIS                            $$ RETRACT TOOL AND INDEX PRIMARY AXIS
CASE/IPRM
        WHEN/1
                        ROTATE/AAXIS,INCR,180
        WHEN/2
                        ROTATE/BAXIS,INCR,180
        WHEN/3
                        ROTATE/CAXIS,INCR,180
ENDCAS
SPINDL/OFF
COOLNT/OFF
STOP
SPINDL/ON                       $$ ADVANCE TOOL BACK TO SAME POS
COOLNT/ON
GODLTA/-RTDIS
IF(ICY.EQ.1) THEN
        CYCLE/ON
ENDIF
DMY=POSTF(21)                           $$ REDO GOTO POINT
DMY=POSTF(13)
TERMAC
```

12 Interactive Debugger

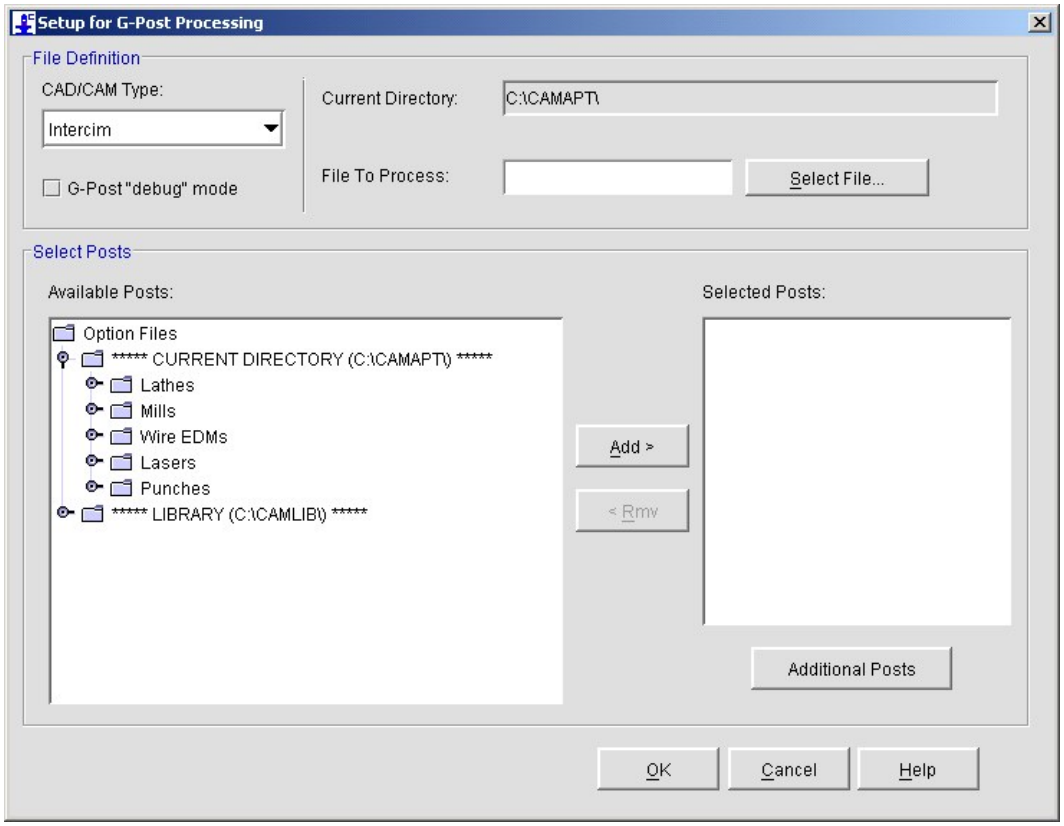
12.1 Introduction:

Note: This feature is only available on the Windows operating systems.

The CimPRO Graphical User Interface must be used to launch the debugger. With OEM versions of G-Post for PTC and Surfcam, you will need the standalone version of CimPRO. This will also allow users to run the G-Post outside the CAD/CAM system.

12.2 The Debug Process:

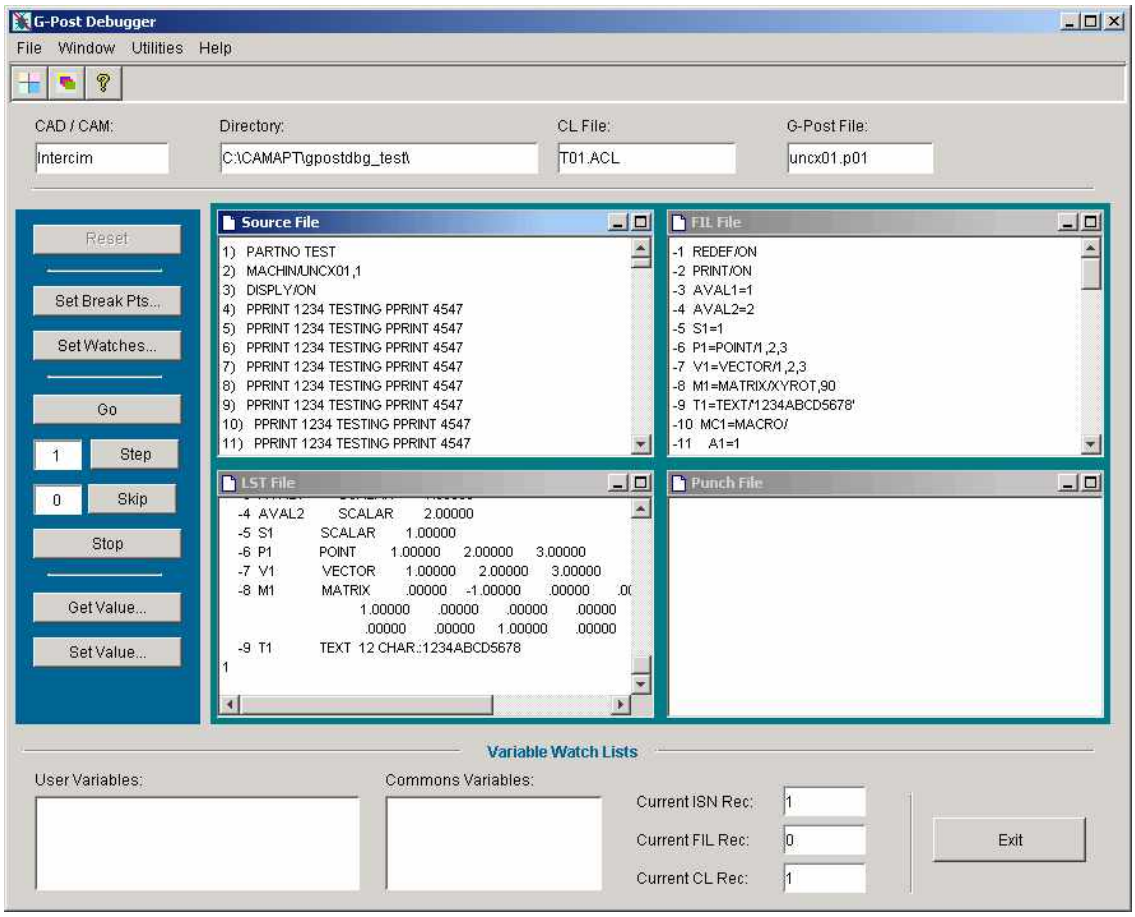
Initial Setup: To use the Interactive Debugger interface (GUI) you must first start CimPRO. On the CimPRO GUI you must press the **CL File thru G-Post** button. On the **Setup for G-Post Processing** screen select your CAD/CAM Type and check the box **G-Post "debug" mode**, select the File to Process and Post Processor then click OK



Note: The Interactive Debugger is not available when processing an APT job. APT users can create an ACL file by processing their APT source then utilizing the **CL File thru G-Post** process type selecting "Intercim" as your CAD/CAM type. We also recommend that UG-APT and CATIA CAD/CAM users create an ACL file to use while running the debugger. These interfaces utilize part of the APT system to process the CL data and utilizing the ACL file will make tracking the ISN numbers in the source file easier.

Once this is done, click the **Debug G-Post** button on the Main CimPRO screen to start the G-Post debug process. A message will appear that the debugger is launching.

The main debugger windows will appear along with the following message box. Click **OK** and you are ready to start debugging your post processor.



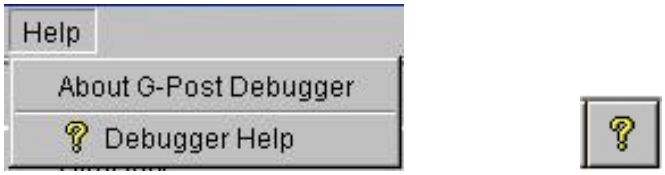
Main G-Post Debugger Window

File Views: The interface will show the **Input Source** file, **FIL** file, **Listing (LST)** file and **Punch** file in four separate windows. As the debugger runs through the process these files are updated interactively in real-time. The user can scroll through these files using the bar and buttons to the right of each window.

Note: The full **Listing (LST)** and **Punch** files are not displayed in their respective windows. Only a portion of these files is displayed since these files can become very large. This large file size consumes huge amounts of physical memory that can lead to poor debugger performance and increased processing time. The user can always view the complete version of these files using their default editor any time the debugger is not actively processing. (i.e. At a break/watch point, stopped or at the end of the CL file)

There is an option in the debugger, this is accessible using the **Window** pull-down menu or the small buttons, to display these windows in either a tiled or cascade view.

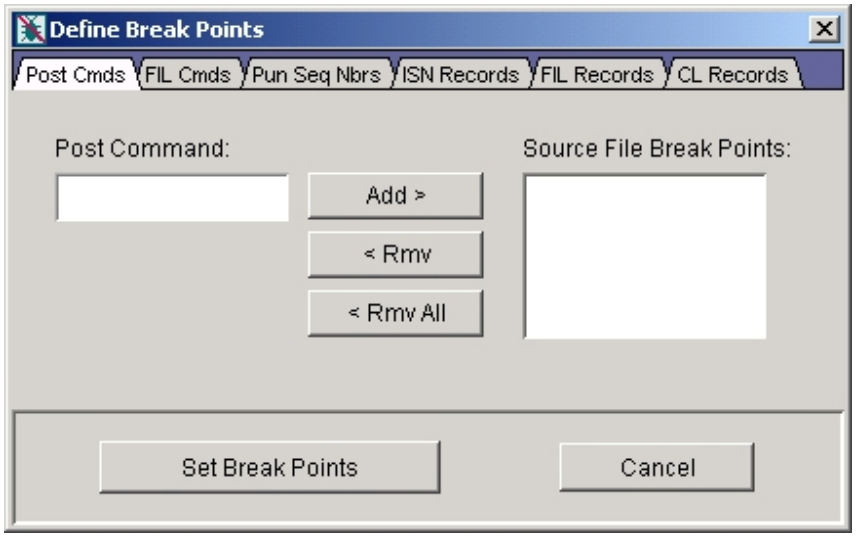
Help: There is a complete help section for the debugger available to the users. The help section is accessible using the **Help** pull-down menu or the small button.



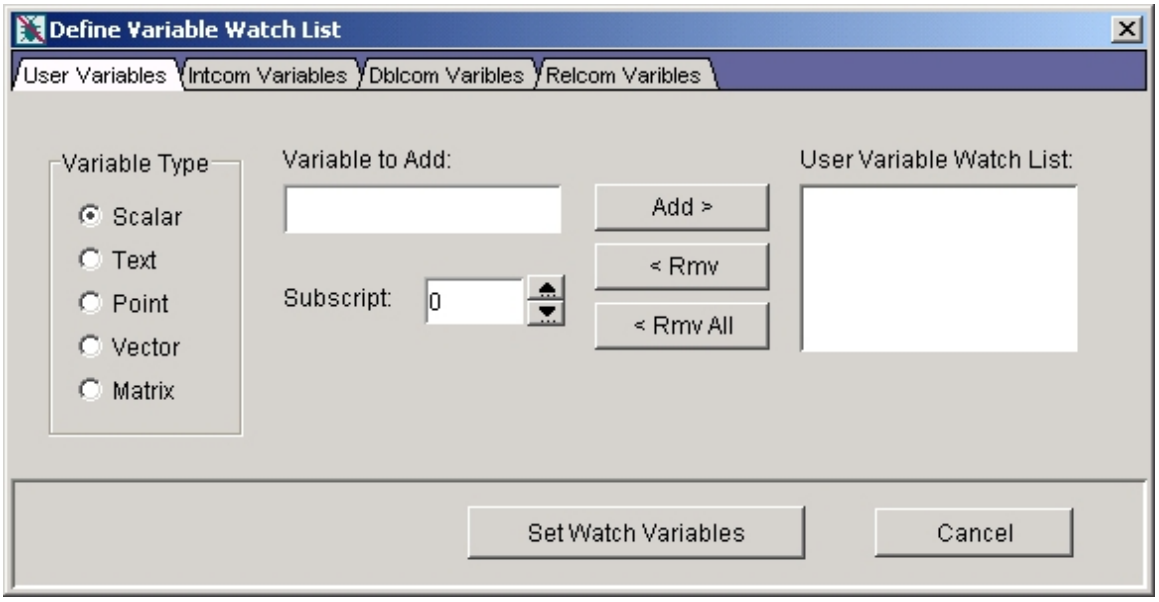
Command Buttons: Once the G-Post is in debug mode, the GUI will allow you to control the debug session with the following commands/buttons.

Reset = Starts the debugging process over after a file is edited.
Set Break Pts = Set a break point for input, FIL commands, etc.
Set Watches = Set a watch for a common location, FIL variable, etc.
Go = Continue processing until next break point or watch is reached
Step = Process a single/many input or FIL record(s)
Skip = Skip a single/many input or FIL record(s)
Stop = Stop the current debug process
Get Value = Get values of common locations or FIL variables
Set Value = Set values of common locations or FIL variables

Setting Command/Record Break Points: You can set a break point at a post command or FIL command like SPINDL or LOADTL. When the G-Post encounters one of these commands, it will return control back to GUI for more commands. You can set a break point at a input record, CL record or FIL record or at a sequence or tape block number. When the G-Post encounters one of these records, it will return control back to UI for more commands



Setting watch points: You can set a watch point for a post common location or a FIL symbol/variable. When the G-Post encounters a change at one of these locations, it will return control back to GUI for more commands.



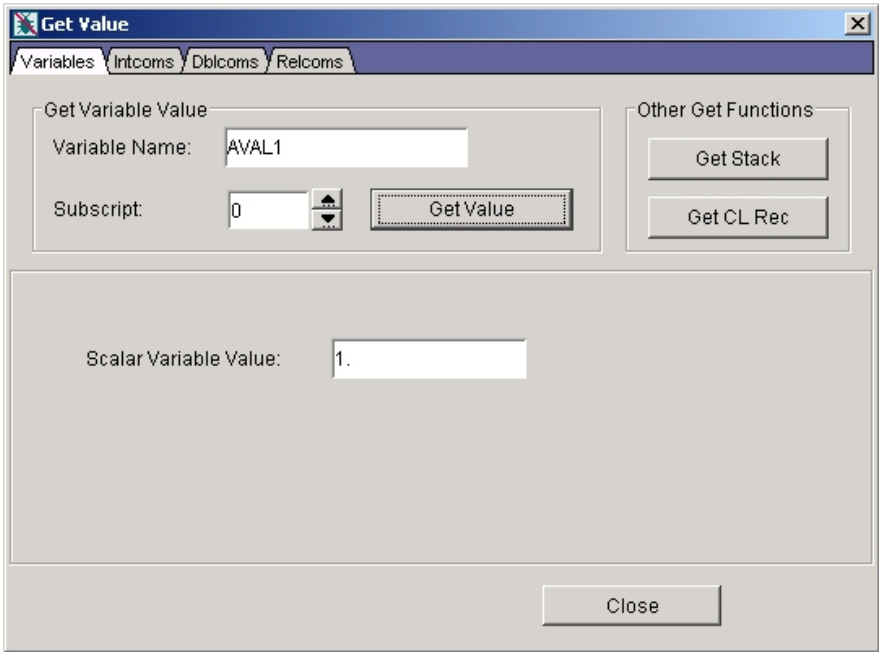
GO Command: Say you have set a set break or set watch point and you want to process the next many records. You can do so with the **Go** button and the G-Post will process until a new break point is found. If you are finished debugging, then you can cancel all break and watch points and hit **Go** to process to the end of the CL file (FIN

Stepping Through Records: You can also execute one or more input source or FIL records with the **Step** button. When the G-Post is done processing the records, it will return control back to UI for more commands.

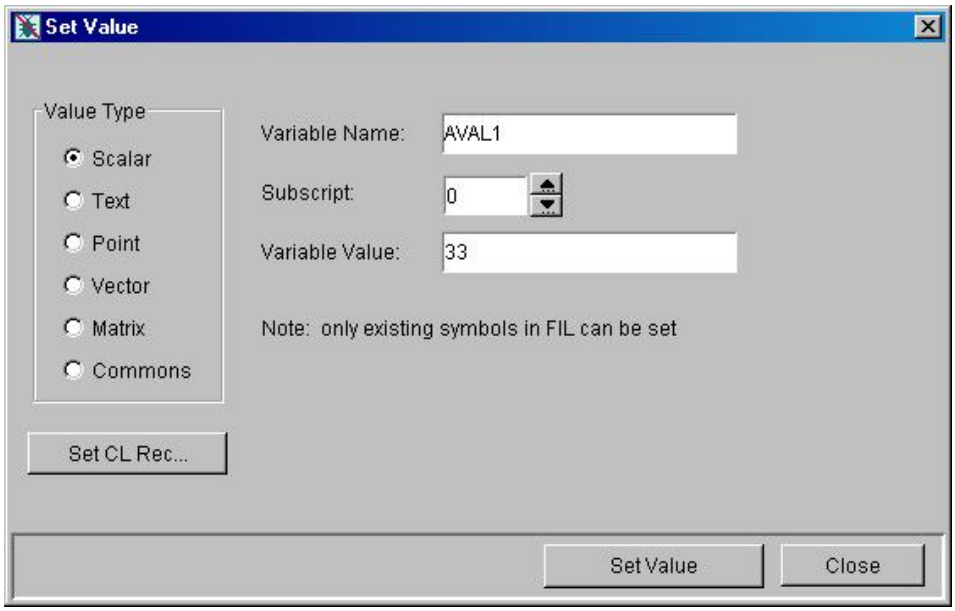
Skipping Records: Once a break point is hit and control is back to the GUI, you can skip one or more records by setting the Skip value and pressing the **Skip** button.

Stop Command: Once the Debugger is running you can stop the process by pressing the **Stop** button. You can then edit your input, option or FIL file and then and restart the process in a new debug cycle.

Getting Common Location or Variable Values: You can examine a value of a common location or FIL variable with the **Get Value** button.



Setting Common Location or Variable Values: You can change or set common location or FIL variable value with the **Set Value** button.

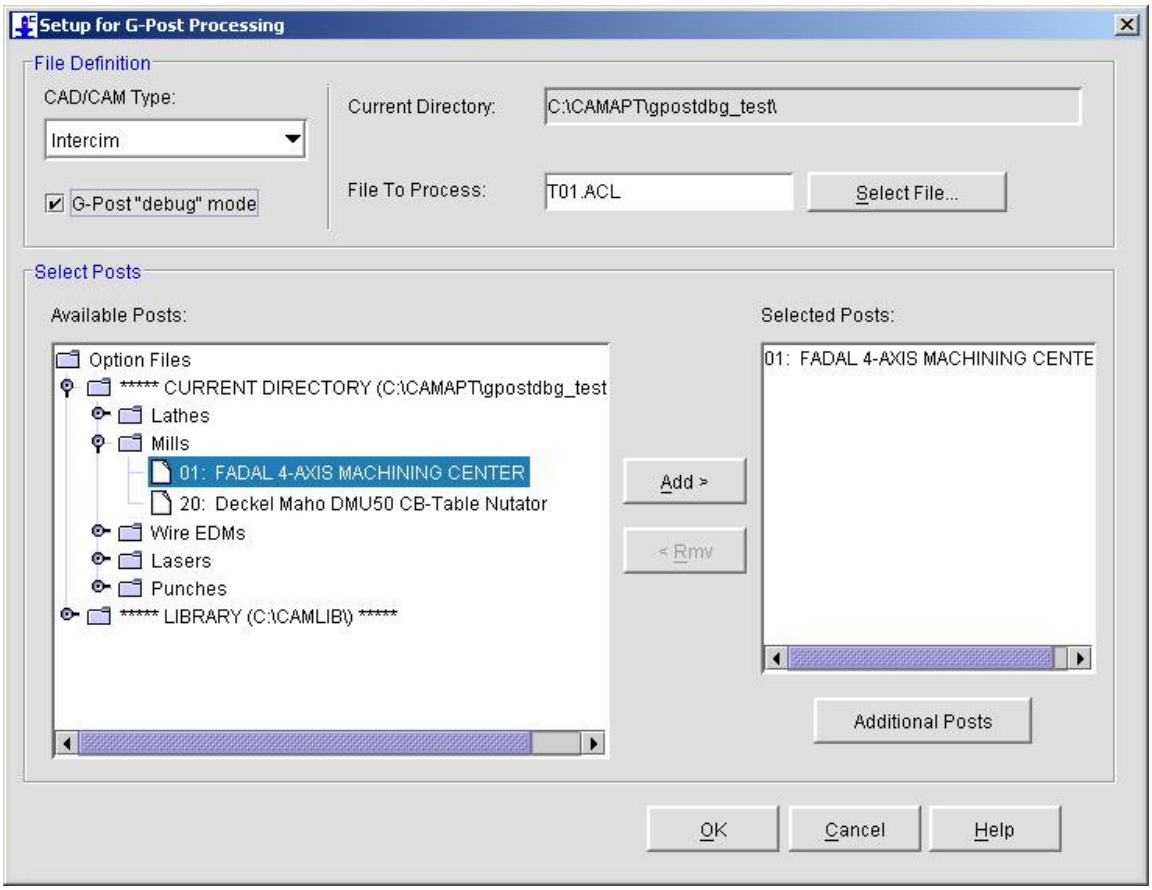


Note: When the G-Post encounters an error in the FIL file (like an undefined variable or bad command, etc), it will return control back to GUI with the file pointer in the FIL file set to the offending line. The pointer in the LST File windows will be set to the line containing the error message. At this stage, you need to STOP the debug process and correct the error (via editing the FIL file) and re-start again using the Start/SETUP button.

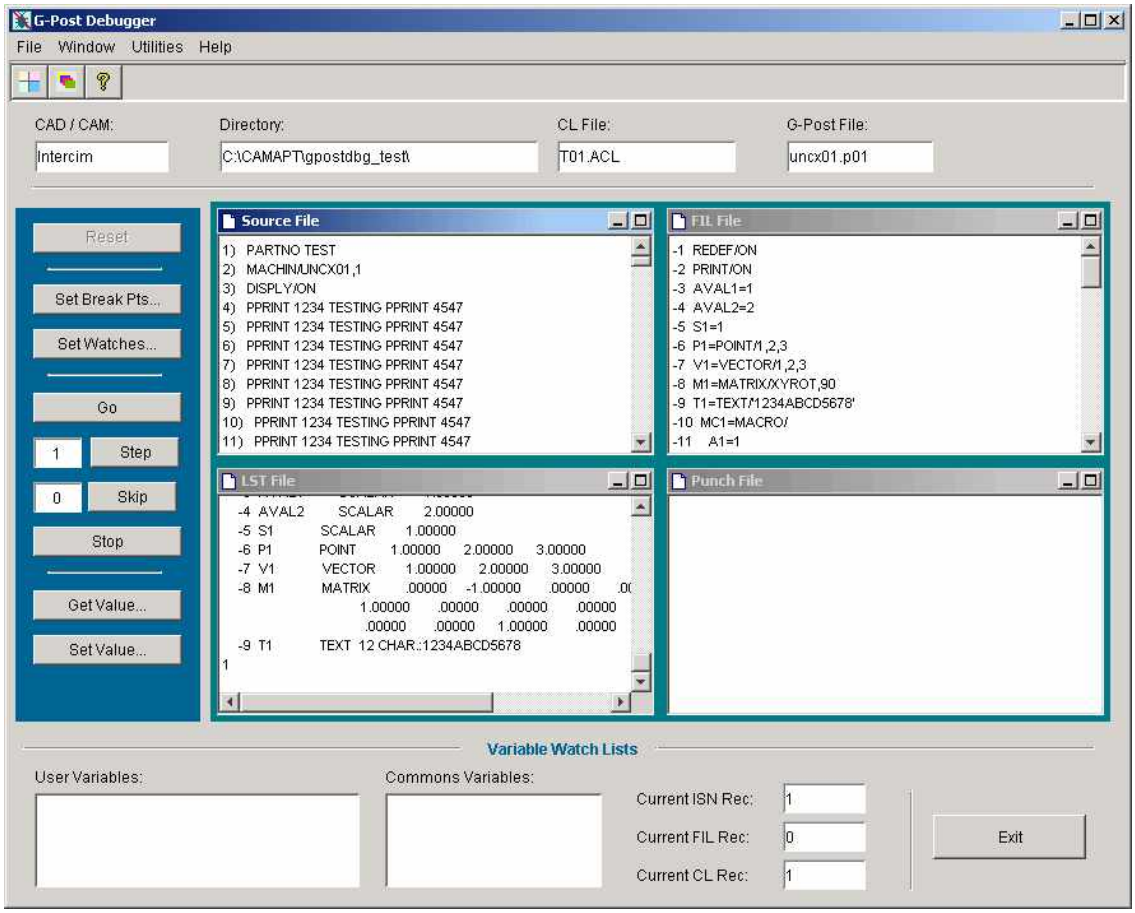
12.3 Example:

For this test run of the Interactive Debugger we are using a simple Austin N.C., Inc. ACL file (T01.ACL) and a fairly basic Fadal 4 axis Mill G-Post (UNCX01.P01 & UNCX01.F01).

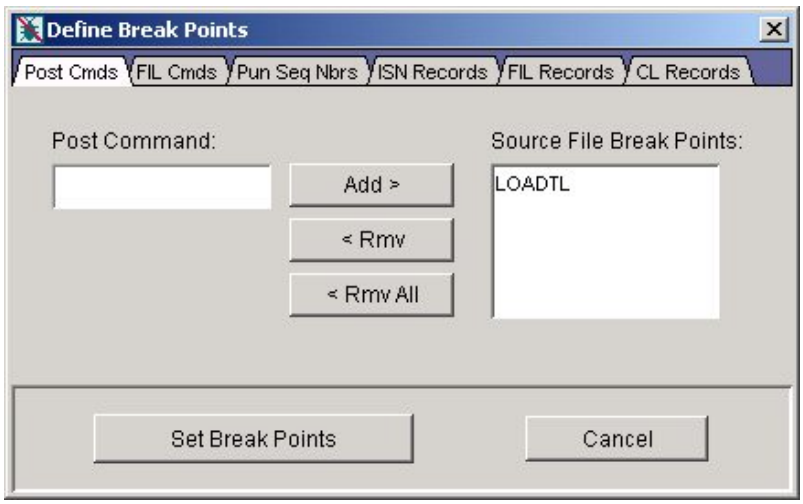
1. Start the CimPRO Graphical User Interface (GUI). On the main screen of the CimPRO GUI press the **CL File thru G-Post** button to configure CimPRO to run the Interactive Debugger. On the **Setup for G-Post Processing** screen select Intercim as the CAD/CAM Type and check the box **G-Post "debug" mode**, select the File to Process "**T01.ACL**" and the post processor to debug "**01 - FADAL 4-AXIS MACHINING CENTER**" then click the **OK** button.



2. From the main screen of the CimPRO GUI press the **Debug G-Post** button to start the debugging process. The "Launching the G-Post Debugger Application" dialog box will appear followed be the "Setup Complete…" dialog box and the main debugger window. Click **OK** in the "Setup Complete…" dialog box, you are now ready to start debugging your post processor.

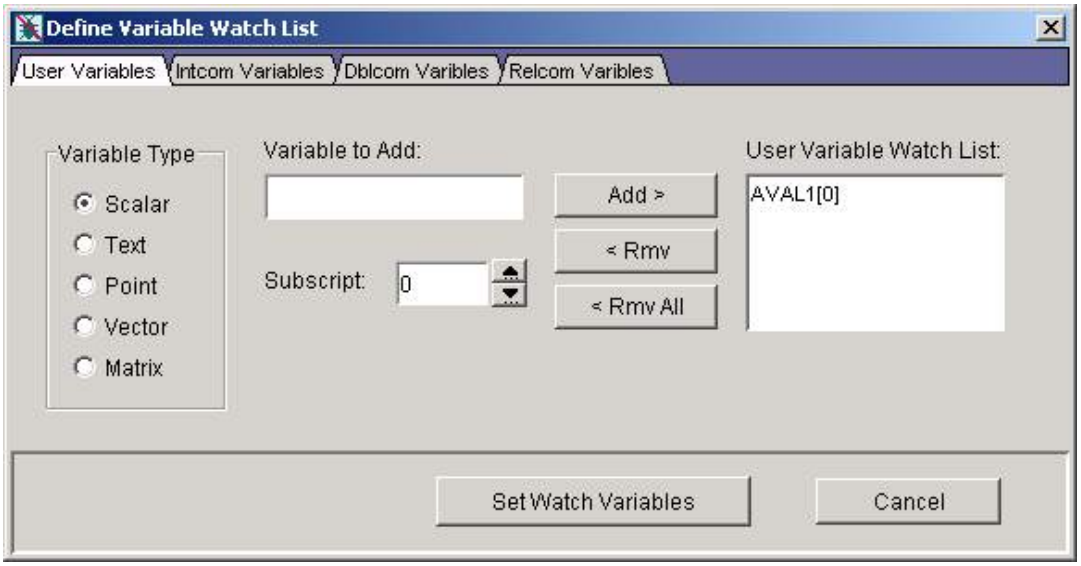


3. To find problems in your post it is helpful to have the post stop at certain points in the debug process. There are several ways to make the debugger stop. All of these choices are contained in the “Define Break Points” dialog box. For this example we are going to set a break point for the debugger to stop when the vocabulary word LOADTL is encountered. Click on the **Set Break Pts** button and the “Define Break Points” dialog box will appear.



Type the word LOADTL into the **Post Command** box and press the **Add>** button. The word LOADTL will appear in the **Source File Break Points:** box. Press the **Set Break Points** button; the debugger is now set to break (Stop) on every occurrence of the word LOADTL in our Input Source file.

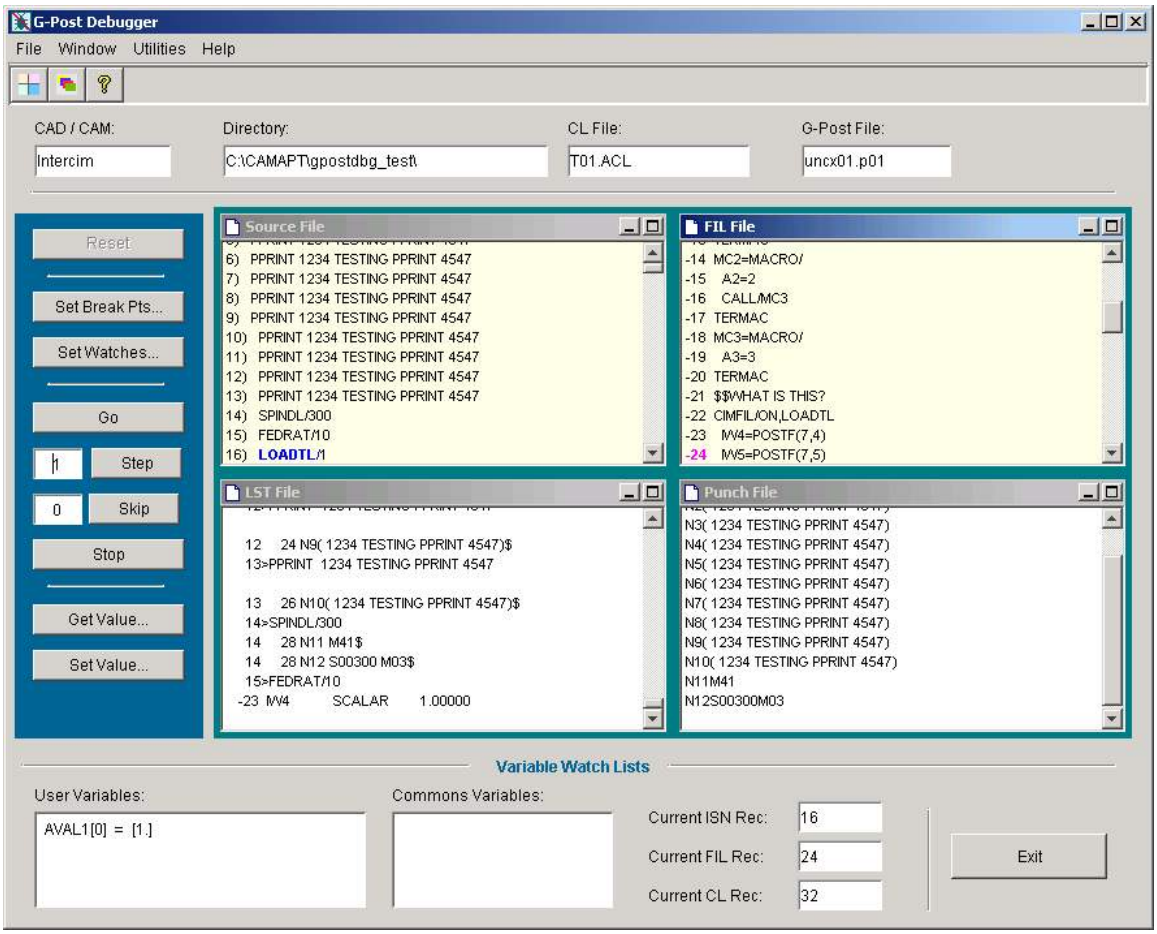
4. Another way to find problems in your post is to have the post stop when certain variables (User, INTOM, RELCOM, DBLCOM) change. To help us debug our post in this example we want to watch a certain User variable. To do this we will add the variable name to the watch list by pressing the **Set Watches** button. The "Define Variable Watch List" dialog box will appear.



Make sure the "User Variables" tab is selected. Select the **Variable Type - Scalar** and type the User variable name "AVAL1" into the **Variable to Add** box and press the **Add>** button. The User variable name "AVAL1" will appear in the **User Variable Watch List** box. Press the **Set Watch Variables** button; the debugger is now set to break (Stop) every time the User variable "AVAL1" gets updated.

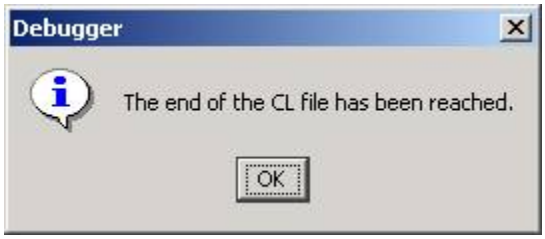
5. Start the debugger process running by pressing the **Go** button. The four (4) files displayed on the main screen will begin to update as the G-Post reads and processes CL records from the Input Source file. Any time the debugger see the User variable "AVAL1" update its value it will stop. At this point it will also display the User variable name and current contents in the **Variable Watch Lists** area at the bottom of the debugger's main windows and the FIL file windows will highlight the User variable in magenta.

6. The debugger will also stop every time it encounters the vocabulary word LOADTL during the debug process. The Source file window will have the line that contains LOADTL/… highlighted in blue. If there is a FIL section to capture LOADTL, and this post has one of these sections, the FIL file windows will display the next line in the FIL file that has become active and it is highlighted in magenta. These events are shown in the next picture:



7. To watch the FIL file as it executes we can process one FIL command at a time using the **Step** button. Enter the number of command(s) you would like to execute at one time in the box next to the **Step** button, for this example we will set it to one (1). Each time we press the **Step** button the next line of our FIL file LOADTL section will execute. The Listing (LST) and Punch files will update after each press of this button allowing us to watch what is happening. We will also see value of the User variable "AVAL1" as it changes.

8. Once we are finished stepping through our FIL code we need to press the **Go** button again. This will cause the debugger to continue processing until another break point is reached. If no further break points are encountered, this is the case with our example, the debugger will display the following message:



9. If you are finished debugging your post you can now close the debugger by pressing the **Exit** button. You can also start the debugging process over by pressing the **Reset** button. If you need to make changes to your Input Source file and/or FIL file you can **only** do that while the debugger is **stopped**. You must reset the debugger by pressing the **Reset** button to get it to recognize these changes. Resetting the debugger causes it to read in the Input Source, FIL and Option files and restart the Listing (LST) and Punch files.

Note: The only two conditions where the debugger is "stopped" are if it has reached the end of the CL file or it the user has pressed the Stop button.

12.4 Frequently Asked Questions (FAQ):

Q: Can I edit input source file after tracing a problem?

A: Yes - First make sure the debugger is stopped. Then use the editor of your choice to edit the file and use the **Reset** button to check output again.

Q: Can I change the Option file after tracing a problem?

A: Yes - First make sure the debugger is stopped. Use Option File Generator to make the changes and use the **Reset** button to check output again.

Q: Can I edit the FIL file after tracing a problem?

A: Yes - First make sure the debugger is stopped. Then use the editor of your choice to edit and use the **Reset** button to check output again.

Q: I get a FIL error in the LST file, how can I find which line in the FIL file is causing this problem?

A: If you are already running the debugger the offending line will be displayed in the LST file window. If not, start the debugger and press the **Go** button, it will automatically stop at the offending FIL record.

Q: How can I stop at a point where a warning is output?

A: Set a watch by pressing the **Set Watches** button and selecting the "Intcom Variables" tab. On this tab enter the number 1932 in the "Intcom to add" box and press the **Add>** button then press **Set Watch Variables** button.. Start the debugger by pressing the **Go** button and it will stop after each warning is output. This happens because Intcom(1932) is updated each time there is a warning output.

Q: I get M81 output on tape block N380, how can I find out which input command is causing this?

A: Set a break point by pressing the **Set Break Pts** button and selecting the "Pun Seq Nbrs" tab (Punch Sequence Numbers). On this tab enter the number 380 in the "Punch File Seq Nbs" box and press the **Add>** button then press the **Set Break Points** button. Start the debugger by pressing the **Go** button and it will stop at the Punch file block before N380. Then you can step one block at a time to see what is causing this output.

Q: I have a general macro to save the current CL record with XX=POSTF(20). This macro is called by many FIL sections and other macros. How can I find the calling sequence of macros at a particular command?

A: Set break point each time input command (ISN Record), select GO and it will stop at this line. Use Get Value / Get Stack buttons to see FIL call sequence.

Q: I get G2/G3 output except on block N500, which generates G1. How can I find out why this is happening?

A: Set break point by pressing the **Set Break Pts** button and selecting the "Pun Seq Nbrs" tab (Punch Sequence Numbers). On this tab enter the number 480, or some number near but before 500, in the "Punch File Seq Nbs" box and press the **Add>** button then press the **Set Break Points** button. Start the debugger by pressing the **Go** button and it will stop at the Punch file block before N480. Step ahead one record at a time using the **Step** button with it's value set to one (1). Once you see the that the pointer in the Source file windows is on the command CIRCLE press the **Get Value** button then the **Get CL Rec** button to see the CIRCLE/TYPE 3000 record's values. Check "WORD(15)", this is the circle radius, and see if this exceeds the maximum radius set in the Option file for this post.

Q: I get an X-axis limit error on input line 200. Can I modify this GOTO (trial) during debug to see if this will remove the limit error?

A: Yes - Set break point by pressing the **Set Break Pts** button and selecting the "ISN Records" tab. On this tab enter the number 200 in the "ISN Record" box and press the **Add>** button then press the **Set Break Points** button. Start the debugger by pressing the **Go** button and it will stop at the Input Source file line number 200. Press the **Set Value** button then the **Set CL Rec** button to change X-location of the GOTO/cmd. You can do this by setting CL Word Number 6. Press the **Go** button again to see the results.

Q: When can I use the **Stop** button?

A: You can click it any time to stop the G-Post's execution and return to the debugger main window. To restart once the debugger has been stopped click the **Reset** button.

Q: I have completed the Debug process. How can I run the G-Post in normal mode?

A: Press the **CL File thru G-Post** button and on the CimPRO "Setup for G-Post Processing" dialog box, uncheck the G-Post "debug" mode check box. Then press the **OK** button to return to the CimPRO main window and press the **Process** button.

Index

_

A

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W


ANC
Austin N.C., Inc.