Customizing CVNC™

CADDS® 5i Release 14

DOC38502-002

Parametric Technology Corporation

GB2366639B 13-October-2004	GB2353376 05-November-2003	5,140,321 18-August-1992
GB2363208 25-August-2004	GB2354686 15-October-2003	5,423,023 05-June-1990
(EP/DE/GB)0812447 26-May-2004	6,545,671 B1 08-April-2003	4,310,615 21-December-1998
GB2365567 10-March-2004	GB2354685B 18-June-2003	4,310,614 30-April-1996

(GB)2388003B 21-January-2004	GB2354683B 04-June-2003	4,310,614 22-April-1999
6,665,569 B1 16-December-2003	6,608,623 B1 19-August-2003	5,297,053 22-March-1994
GB2353115 10-December-2003	6,473,673 B1 29-October-2002	5,513,316 30-April-1996
6,625,607 B1 23-September-2003	GB2354683B 04-June-2003	5,689,711 18-November-1997
6,580,428 B1 17-June-2003	6,447,223 B1 10-September-2002	5,506,950 09-April-1996
GB2354684B 02-July-2003	6,308,144 23-October-2001	5,428,772 27-June-1995
GB2384125 15-October-2003	5,680,523 21-October-1997	5,850,535 15-December-1998
GB2354096 12-November-2003	5,838,331 17-November-1998	5,557,176 09-November-1996
GB2354924 24-September-2003	4,956,771 11-September-1990	5,561,747 01-October-1996
6,608,623 B1 19-August-2003	5,058,000 15-October-1991	(EP)0240557 02-October-1986

Table of Contents

Pass-through Statements and Macros

Preface

Customizing CVNC provides an overview of and instructions for customizing CVNC with command and execute files, status definition files, pass-through statements, system variables, and macros.

Related Documents

The following documents may be helpful as you use *Customizing CVNC*:

- *Understanding CVNC*
- *CVNC System Guide and Menu Reference*
- *CVNC Editor Guide*
- Command reference and user guide(s) for your application
- *CVNC Master Index*
- *CVNC System Variables Guide*
- *CVMAC Language Reference*

Book Conventions

The following table illustrates and explains conventions used in writing about CADDS applications.

Convention	Example	Explanation
Menu selections and options	List Section option, Specify Layer field	Indicates a selection you must make from a menu or property sheet or a text field that you must fill in.
User-selected graphic location	X, d_1 or P1	Marks a location or entity selection in graphic examples.

User input in CADDS text fields and on any command line	`cvaec.hd.data.param`
	`tar -xvf /dev/rst0`
		Enter the text in a CADDS text field or on any command line.

System output	`Binary transfer complete.`	Indicates system responses in the CADDS text window or on any command line.
Variable in user input	`tar -cvf /dev/rst0` filename	Replace the variable with an appropriate substitute; for example, replace filename with an actual file name.
Variable in text	tagname	Indicates a variable that requires an appropriate substitute when used in a real operation; for example, replace tagname with an actual tag name.
CADDS commands and modifiers	INSERT LINE TANTO	Shows CADDS commands and modifiers as they appear in the command line interface.
Text string	`"SRFGROUPA"` or `'SRFGROUPA'`	Shows text strings. You must enclose text string with single or double quotation marks.
Integer	n	Supply an integer for the *n.*
Real number	x	Supply a real number for the *x.*
#	`# mkdir /cdrom`	Indicates the root (superuser) prompt on command lines.
%	`% rlogin` remote_system_name `-l root`	Indicates the C shell prompt on command lines.
$	`$ rlogin` remote_system_name `-l root`	Indicates the Bourne shell prompt on command lines.

Window Managers and the User Interface

According to the window manager that you use, the look and feel of the user interface in CADDS can change. Refer to the following table:

Look and Feel of User Interface Elements

User Interface Element	Common Desktop Environment (CDE) on Solaris, HP, and IBM	Window Manager Other Than CDE on Solaris, HP, IBM, and Windows
Option button	ON — Round, filled in the center OFF — Round, empty	ON — Diamond, filled OFF — Diamond, empty
Toggle key	ON — Square with a check mark OFF — Square, empty	ON — Square, filled OFF — Square, empty

Online User Documentation

Online documentation for each book is provided in HTML if the documentation CD-ROM is installed. You can view the online documentation in the following ways:

- From an HTML browser
- From the Information Access button on the CADDS desktop or the Local Data Manager (LDM)

Please note: The LDM is valid only for standalone CADDS.

You can also view the online documentation directly from the CD-ROM without installing it.

From an HTML Browser:

1. Navigate to the directory where the documents are installed. For example,

   `/usr/apl/cadds/data/html/htmldoc/` (UNIX)

   Drive:`\usr\apl\cadds\data\html\htmldoc\` (Windows)
2. Click `mainmenu.html`. A list of available CADDS documentation appears.
3. Click the book title you want to view.

From the Information Access Button on the CADDS Desktop or LDM:

1. Start CADDS.
2. Choose Information Access, the i button, in the top-left corner of the CADDS desktop or the LDM.
3. Choose DOCUMENTATION. A list of available CADDS documentation appears.
4. Click the book title you want to view.

From the Documentation CD-ROM:

1. Mount the documentation CD-ROM.
2. Point your browser to:

   CDROM_mount_point`/htmldoc/mainmenu.html` (UNIX)

   CDROM_Drive:`\htmldoc\mainmenu.html` (Windows)

Online Command Help

You can view the online command help directly from the CADDS desktop in the following ways:

- From the Information Access button on the CADDS desktop or the LDM
- From the command line

From the Information Access Button on the CADDS Desktop or LDM:

1. Start CADDS.
2. Choose Information Access, the i button, in the top-left corner of the CADDS desktop or the LDM.
3. Choose COMMAND HELP. The Command Help property sheet opens displaying a list of verb-noun combinations of commands.

From the Command Line: Type the exclamation mark (!) to display online documentation before typing the verb-noun combination as follows:

```
#01#!INSERT LINE
```

Printing Documentation

A PDF (Portable Document Format) file is included on the CD-ROM for each online book. See the first page of each online book for the document number referenced in the PDF file name. Check with your system administrator if you need more information.

You must have Acrobat Reader installed to view and print PDF files.

The default documentation directories are:

- `/usr/apl/cadds/data/html/pdf/`doc_number`.pdf` (UNIX)
- CDROM_Drive`:\usr\apl\cadds\data\html\pdf\`doc_number`.pdf` (Windows)

Resources and Services

For resources and services to help you with PTC (Parametric Technology Corporation) software products, see the *PTC Customer Service Guide*. It includes instructions for using the World Wide Web or fax transmissions for customer support.

Documentation Comments

PTC welcomes your suggestions and comments. You can send feedback electronically to `doc-webhelp@ptc.com`.

Chapter 1

Overview of CVNC Customization

This chapter describes the various methods of customizing CVNC.

- How CVNC Can Be Customized
- Prerequisites for CVNC Customization
- Files Used for Customization
- Searching for and Creating Files
- Overriding the CVPATH Variable
- Tailoring Your .caddsrc File

How CVNC Can Be Customized

You can customize CVNC with

- Command, execute, and status definition files
- Pass-through statements
- User-definable system variables
- CVMAC macros
- Online documentation

The language of CVNC is contained in a pair of binary files called grammar files. These files define CVNC commands, macros, and pass-through statements. Grammar files can be modified or added to. This capability allows you to customize CVNC and its language.

Command, Execute, and Status Definition Files

Command, execute, and status definition files can be used to customize CVNC.

Command and Execute Files

Command and execute files are text files containing a sequence of commands. They are useful for storing repetitive functions such as configuring the machine or changing the tool. For more information, see the *CVNC Editor Guide,* and the *CVNC System User Guide and Menu Reference.*

- Command files - are accessed from the Job Control File (JCF) with the CALL command. A command file must contain only CVNC commands. These commands are executed, but only the CALL command becomes a visible part of the JCF.
- Execute files - are accessed from the JCF with the EXEC command. An execute file can contain CVNC commands or a combination of CVNC and CADDS commands (using the ~ (tilde), RESU, or IC commands [SYS] to enter the CADDS environment). The contents are executed and the CVNC commands become a permanent part of the JCF.

Status Definition Files

You can continuously display the values of system variables in the status display area of your screen. To define a status display, you can create a status definition file at the operating system level. This file specifies the system variables you want to display, as well as the format in which you want to display them. For more information, see the *CVNC System User Guide and Menu Reference.*

Pass-through Statements

Pass-through statements are statements passed directly to the output file; they do not affect CVNC processing. These statements control output processing or the machine tool. You can add new pass-through statements to the grammar file. You can also modify the supplied set of statements for APT, CLFile, and COMPACT II output formats.

Setting System Variables in Pass-through Statements

You can create a pass-through statement that sets user-definable system variables. There are three types of user-definable system variables: text, numeric, and coordinate. You can reference these variables from within a macro, display the current status of the variable in the status display window, or print the current status in the text window using SHOW or PRINT [SYS].

CVMAC Macros

CVNC macros are procedures written in the high-level language CVMAC. Macros may contain logic to build intelligence into the system. A macro, for example, may perform a set of moves at a specified depth and then step down several times to repeat these moves until you reach the depth required. CVNC supports four types of macros:

- Command macros; are called from CVNC like any CVNC command. CVNC supplies several command macros. You can modify these to suit your needs.
- Check macros; a set of procedures for CVNC commands, are executed after a command is processed. They check to see that the executed command meets certain criteria. For example, a check macro could verify spindle retraction from the part before indexing a rotary table.
- Output macros; create customized output for CVNC by passing machine control statements directly to output.
- Point macros; are accessed at predefined points during command execution. They can be used with the AREAMILL [M2], SURFCUT3 [M3], SURFCUT4 [M5 Four Axis], and SURFCUT5 [M5] commands. You can add machine control statements within tool paths by issuing pass-through statements to the output.

When you create command macros, you must append a statement describing the macro syntax to the grammar files. When you create output or point macros, you may have to append the pass-through statements that these macros output.

Online Documentation

CVNC provides online documentation files for commands, system- supplied command macros, and system variables. You can edit these files to add information relevant to your shop. You can also create new documentation for user-written macros and pass-through statements.

Prerequisites for CVNC Customization

You must meet the prerequisites outlined in this chart before customizing CVNC as described in the remainder of this document. The list that follows the chart explains these prerequisites and refers you to related documentation.

Pre-requisite / Method of Customizing	Ability to use SunOS editor	CVMAC compiler and linker	Ability to program in CVMAC	Under-standing of CVMAC/ CVNC interface	Under-standing of CVNC grammar files	Familiarity with the postprocessor used in your shop	One of the milling products (M2, M3, and M5)
Pass-through Statements	✓				✓	✓	✓
User-definable System Variables	✓				✓		✓
Command Macros	✓	✓	✓	✓	✓		✓
Check Macros	✓	✓	✓	✓			✓
Output Macros	✓	✓	✓	✓	✓	✓	✓
Point Macros	✓	✓	✓	✓	✓		✓
Online Documentation	✓						✓

- You can use any editor that can create a text file. For example, the “vi” editor or dtpad text editor on the SunOS platform.
- CVMAC compiler and linker — Indicates that you have software that compiles and links CVMAC macros.
  - For more information on CVMAC, see the *CVMAC Language Reference.*
- Ability to program in CVMAC — Indicates that you understand how to write CVMAC macros.
- Understanding of CVMAC/CVNC interface — Indicates that you know how to use the CVMAC statements, APLSYSR and APLINF, to read data from CVNC.
- Understanding of CVNC grammar files — Indicates that you know what CVNC grammar files are and what they do. It also means that you can construct grammar file statements and append them to the grammar files.

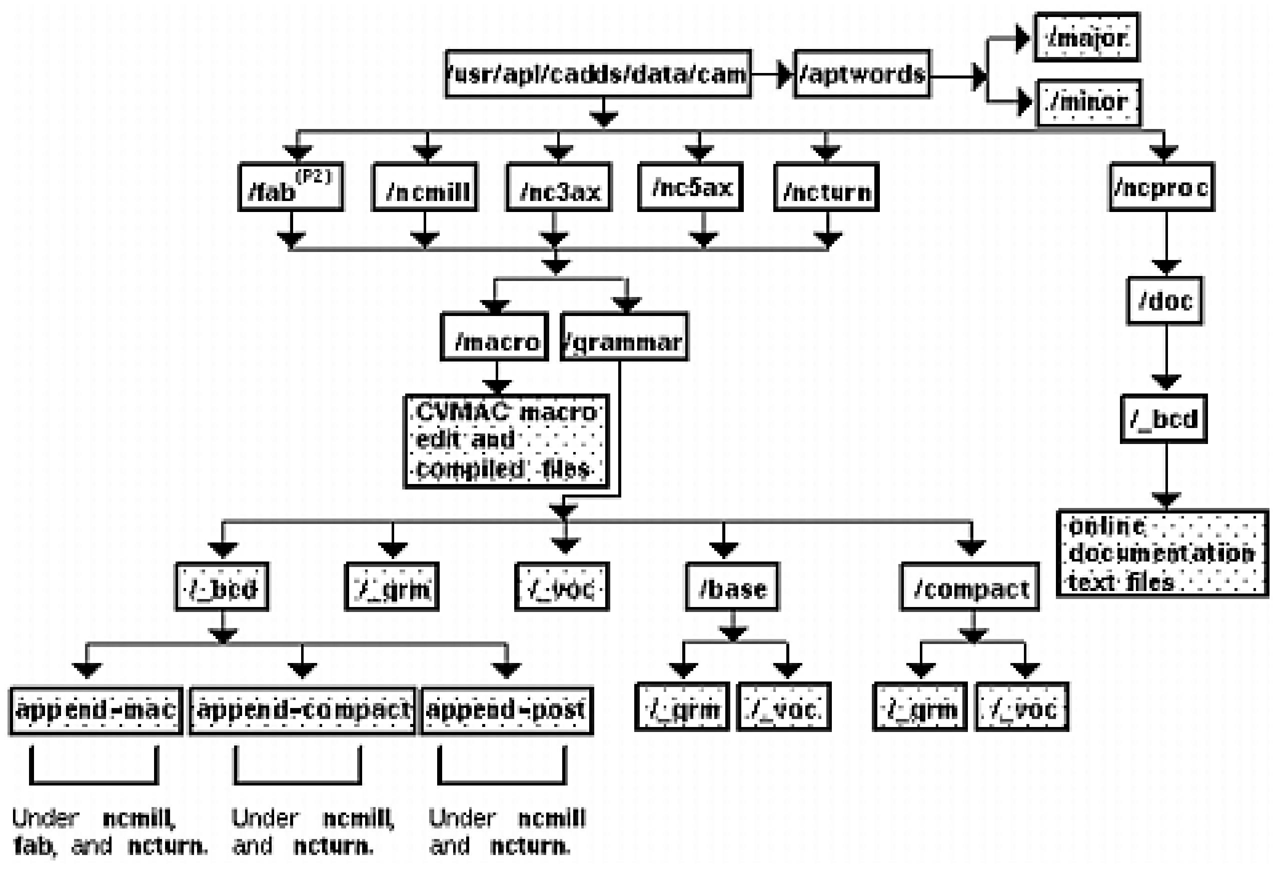
- Familiarity with the postprocessor used in your shop — Indicates that you know what output format your postprocessors or output processors read, and what pass-through statements they require.
- One of the CVNC milling products — Indicates that you can run CVNC-M2, CVNC-M3, or CVNC-M5 on your system.

Files Used for Customization

Different customizing methods require working with different CVNC files. You should be familiar with the location of these files.

The flowchart shows the hierarchy of the files you will append to or modify during customization (the files are inside the shaded blocks).

Figure 1-1 Customization Files



Files Used with Pass-through Statements

When customizing with pass-through statements, you may work with the following files:

- Grammar files- A pair of binary files (`_grm` and `_voc`) containing the language for CVNC. Whether you are adding a new pass-through statement or modifying an existing pass-through statement, you must append to the grammar files.

- Append text files - These files contain CVNC-supplied pass-through statements. To modify these statements, you must edit the appropriate append text file: `append-post` (for APT and CLFile statements) or `append-compact` (for COMPACT II statements). These files are available with CVNC-M2, CVNC-T2, CVNC-P2, and CVNC-PM3 only. CVNC-M2 pass-through statements are available when using CVNC-M3 and CVNC-M5.
- APT/CLFile word files -These files contain all the CVNC-supported APT/CLFile major and minor words. If you are generating CLFile output, and add an APT/CLFile word to a new or existing pass-through statement that is not in the APT/CLFile word files, you must edit the files to include the new word and its integer code.

Files Used with User-written Macros

When customizing with user-written macros, you may work with the following CVNC files:

- Grammar files — This pair of binary files (`_grm` and `_voc`) contains the language for CVNC. To add a new macro, you must append to the grammar files.
- Append text files — This file, `append-mac`, contains grammar file statements for CVNC-supplied macros. You can add grammar file statements for user-written macros to this file.

Files Used with CVNC-supplied Macros

When customizing with CVNC-supplied macros, you may work with the following files:

- Grammar files — These are a pair of binary files (`_grm` and `_voc`) containing the language for CVNC. If you modify the append text file for a supplied macro, you must append to the grammar files.
- Macro text files — These are the CVMAC text files for CVNC-supplied macros. If you want to modify one of these macros, you must edit the appropriate macro text file. These files are available under CVNC-M2, CVNC-T2, and CVNC-P2 directories only. CVNC-M2 macros are accessible when using CVNC-M3 and CVNC-M5.
- Append text files — This file, `append-mac`, contains grammar file statements for CVNC-supplied macros. If you modify the syntax of a CVNC-supplied macro, edit `append-mac` and append it to the grammar.

Files Used with Online Documentation

To customize existing online documentation files, you need to edit the appropriate text file located under

```
/usr/apl/cadds/data/cam/ncproc/doc/_bcd.
```

Searching for and Creating Files

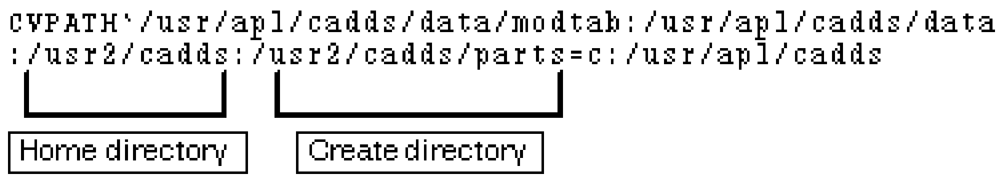
If is important to know where customization commands search for and create files. You will use three main commands for customization. Enter each at the operating system level.

- `ncgram append -` appends text files containing grammar file statements to the CVNC grammar files.
- `cvmcomp -` compiles macros.
- `cvmlink` - links macros.

CVPATH Variable

The CVPATH variable is an environment variable in your `.caddsrc` file. The `.caddsrc` file is a text file in your home directory that contains the information needed when your CADDS session is created. The CVPATH variable defines where `ncgram append`, `cvmlink`, and `cvmcomp` search for and create files.

Example: The CVPATH variable may look like this



```
CVPATH`/usr/apl/cadds/data/modtab:/usr/apl/cadds/data
:/usr2/cadds:/usr2/cadds/parts=c:/usr/apl/cadds
```

The create directory is `/usr2/cadds/parts.` A create directory is specified by following the path with `=c`. If you do not follow any path with `=c`, CADDS will consider the first directory in the path to be the create directory.

In this example, `ncgram append, cvmlink,` and `cvmcomp` search for files in

```
/usr/apl/cadds/data/modtab
/usr/apl/cadds/data
/usr2/cadds
/usr2/cadds/parts
/usr/apl/cadds
```

`ncgram append, cvmlink`, and `cvmcomp` create files in the create directory. For example, if you use `ncgram append` to append a file to a grammar located in `/usr2/cadds/my/grammar, ncgram append` will create the appended grammar in `/usr2/cadds/parts/my/grammar.`

Overriding the CVPATH Variable

When using `ncgram append, cvmcomp,` and `cvmlink`, you can temporarily override the CVPATH variable and search for or create files in other directories. To do this, enter an equal sign (=) followed by the complete path name of the file. For example,

```
%cvmcomp =usr.alison.macro.areamill
```

Please note: File naming conventions are different for CADDS/UNIX commands and UNIX commands. For example, the following cd command changes to the same directory referenced in the cvmcomp command above

```
%cd /usr/alison/macro/areamill
```

In the cvmcomp command, the equals sign (=) replaces the initial slash (/), and a period (.) replaces every following /. For a complete list of file name character conversions that must be used when specifying files for ncgram append, cvmlink, and cvmcomp, see Appendix B, “CADDS/UNIX File Name Character Conventions.”

Tailoring Your .caddsrc File

Now that you know where customization commands search for and create files, you may want to tailor your `.caddsrc` file. For example, you can define separate environment variables for the home directory and the create directory. If the following variables are set:

```
setenv HOME "/usr/alison"
setenv MYCREATE "$HOME/create"
setenv CVPATH "${MYCREATE}:${HOME}:/usr/apl/cadds/data"
```

then files will be created in `/usr/alison/create`. The create directory is equal to MYCREATE.

You can tailor MYCREATE and CVPATH environment variables to create or search for files in different directories, depending on the process being run. Do this by placing conditional statements in your `.caddsrc`.

The screen below shows a `.caddsrc` that creates

- Grammar files in `/usr/alison/grammar`
- CVMAC macro libraries in `/usr/alison/macro`
- CADDS parts in `/usr/alison/create`

For more information on the `.caddsrc` file and a complete list of environment variables, see *Managing CADDS 5i* and *CVware.*

Tailoring Your .caddsrc File

```
setenv CMD /usr/bin/basename $0
if ( $CMD == 'ncgram' ) then
       setenv MYCREATE "$HOME/grammar"
else if ( $CMD == 'cvmcomp' || $CMD == 'cvmlink' ) then
       setenv MYCREATE "$HOME/macro"
else
       setenv MYCREATE "$HOME/create"
endif
setenv CVPATH "${MYCREATE}:${HOME}:/usr/apl/cadds/data"
```

Warning

The following commands run your .caddsrc file. If this file contains the following line

```
      rm -rf /usr/tmp/`whoami`.`hostname`.*
```

the temporary files for any running CADDS process will be deleted. This causes CADDS to fail.

- checkutil
- cvmcomp
- cvmlink
- macutil
- ncgram append
- ncoutput
- ncverify
- printclt

Chapter 2

Using CVNC Grammar Files

This chapter explains how to modify CVNC grammar files to include macros and pass-through statements that you have written.

- %LIMIT
- %VALID
- %INVALID
- %MACRO
- %VMACRO (Variable Argument Macro)
- %HIDDEN
- Special Characters
- Using Metasymbols
- Restrictions on Grammar File Contents
- Modifying the Grammar
- Using the ncgram append Command
- Using a Different Grammar
- Configuring CVNC to Change Grammars
- Using the LANGUAGE Command to Change Grammars

CVNC Grammar Files

CVNC is a language-based system driven by commands, macros, and pass-through statements comprising keywords, symbols, and special characters following a defined syntax. CVNC language is defined in a pair of binary files called its grammar. Each product has its own grammar.

One grammar file of the pair is the vocabulary file (/`_voc`). It contains correct keywords. The other grammar file (/`_grm`) contains correct syntax. CVNC validates input commands by referring to these grammar files and immediately catches misspellings or syntax errors.

CVNC grammar files can be customized. You can modify grammar files to include macros and pass-through statements that you have written. You can also customize CVNC-supplied macros and pass-through statements for your site-specific needs.

Location of Grammar Files

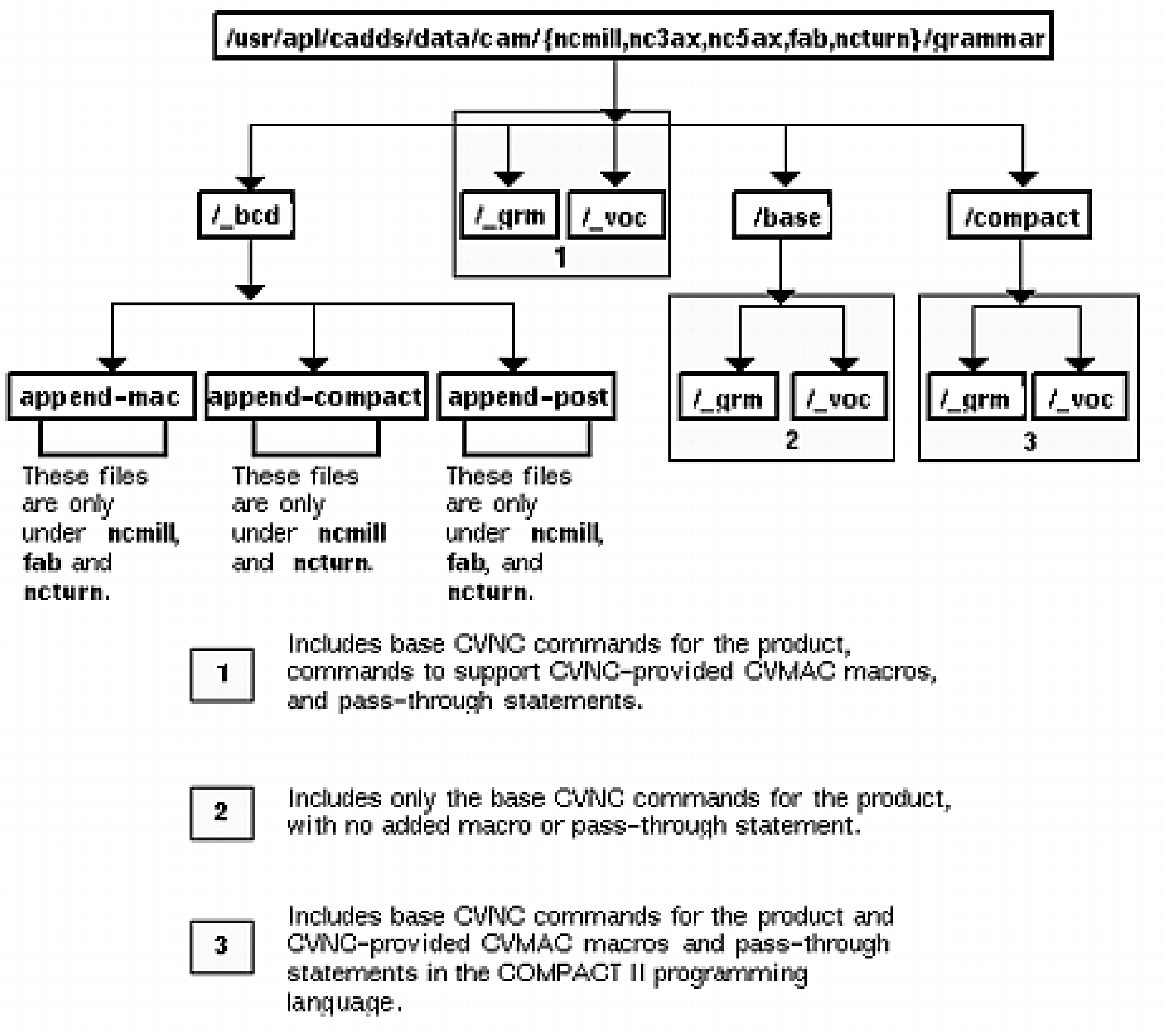
Grammar files, except the rougher grammar file, are located in the directory `/usr/apl/cadds/data/cam.` Under the directory is a directory for each CVNC product. Each product has its own set of three binary grammars:

1. One grammar includes base CVNC commands for the product, commands to support a set of CVNC-provided CVMAC macros, and pass-through statements for CLFile and APT output. This is the default grammar automatically activated upon CVNC entry.
2. A second grammar includes base CVNC commands for the product, commands to support a set of CVNC-provided CVMAC macros, and pass-through statements for COMPACT II output.
3. A third grammar includes only the base CVNC commands for the product, with no added macro or pass-through statements.

Under the `grammar/_bcd` directory for several of the CVNC applications (see the chart, Grammar File Directories, on the next page) are three text files: append-mac, append-post, and append-compact. These text files contain the source grammar for CVNC-supplied macros, APT and CLFile pass-through statements, and COMPACT II pass-through statements. You can modify these files and append them to the grammar. This procedure is described in the chapters on customizing with macros (Chapter 4, “Customizing with CVMAC Macros”) and customizing with pass-through statements (Chapter 3, “Customizing with Pass-through Statements”).

Please note: The CVNC_M3 rougher grammar file is located in the directory /cadds/data/cam/model/grammar. Invoke this grammar file with the MODEL command, and disable it with MODEL OFF.

Figure 2-1 Grammar File Directories


/usr/apl/cadds/data/cam/{ncmill,nc3ax,nc5ax,fab,ncturn}/grammar
/_bcd
/_grm
/_voc
1
/base
/compact
/_grm
/_voc
2
/_grm
/_voc
3
append-mac
append-compact
append-post
These files are only under ncmill, fab and ncturn.
These files are only under ncmill and ncturn.
These files are only under ncmill, fab, and ncturn.
1
Includes base CVNC commands for the product, commands to support CVNC-provided CVMAC macros, and pass-through statements.
2
Includes only the base CVNC commands for the product, with no added macro or pass-through statement.
3
Includes base CVNC commands for the product and CVNC-provided CVMAC macros and pass-through statements in the COMPACT II programming language.

Grammar File Statements

To customize CVNC with macros and pass-through statements, you need to write grammar file statements. Grammar file statements consist of phrases and metasymbols. The following table defines these and related terms, which are described below.

Table 2-1 Grammar File Terminology

Term	Definition
Phrase	Rules and syntax. Consists of one or more of the following: major word, minor words, symbols, and special characters.
Metasymbol	Reusable, named phrase or group of phrases.
Major word	Keyword representing a main purpose or action.
Minor word	Keyword that qualifies a major word.
Symbol	Conditional expression; begins with %.
Special character	Equal sign (=), comma (,), and square brackets ([ ]).
Statement	Consists of phrases and metasymbols; defines the syntax for macros and pass-through statements.

Phrases

Phrases provide rules and syntax for the CVNC language. They are constructed with one or more of the following:

- Major and minor words: Major words represent a main purpose or action. A command always begins with a major word. Minor words qualify a major word (that is, MOVE is a major word; HOME, a minor).
- Symbols: used with major and minor words, express conditions and rules. For example, %CHOOSE is used a minor word.
- Special characters: such as the equal sign (=), can be included in a phrase so that it becomes part of a command syntax.

Metasymbols

A metasymbol is a name given to a commonly used phrase or group of phrases. Metasymbols allow you to replace long phrases and sequences of phrases with just one symbol.

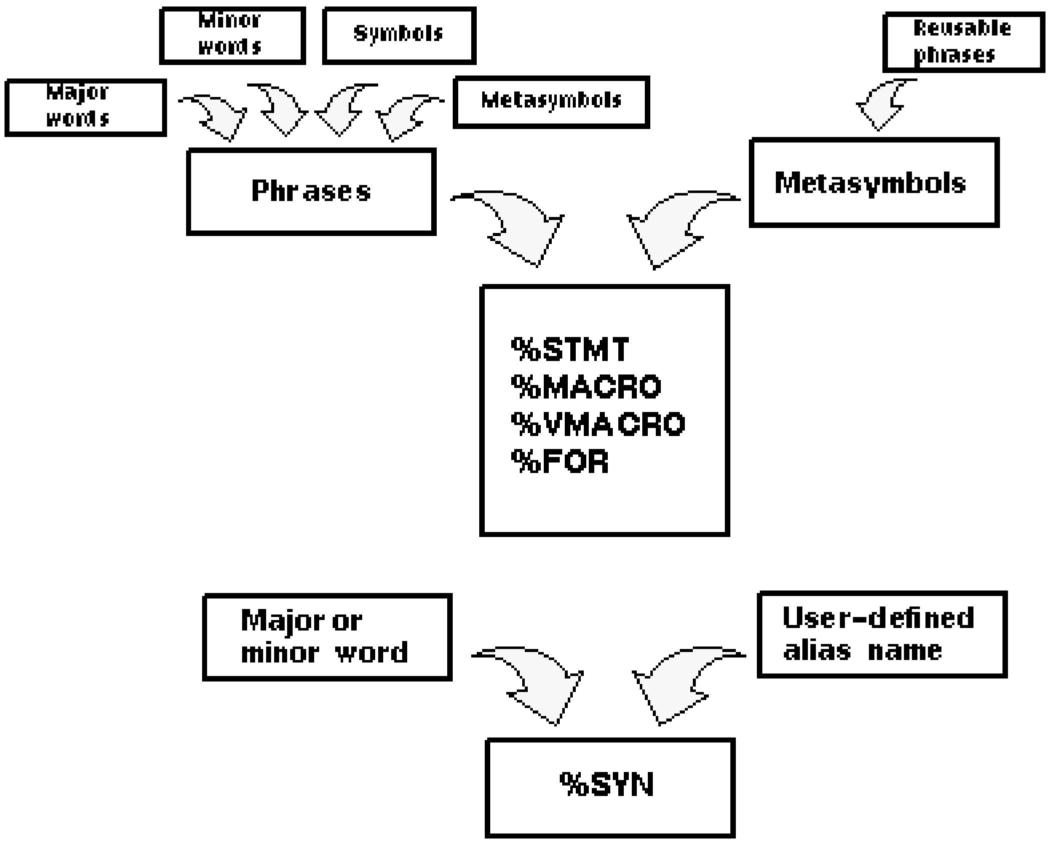
Types and Parts of Statements

Grammar files include four types of statements. The following figure shows the possible parts of each statement type.

- Statements for pass-through statements: These statements all begin with the symbol %STMT, followed by one or more phrases and metasymbols.
- Statements for CVMAC macros: These statements begin with the symbol %MACRO or %VMACRO, followed by one or more phrases.
- Statements defining metasymbols: These statements begin with the symbol %FOR, followed by one or more phrases.
- Statements defining synonyms: Synonyms are aliases given to any CVNC major word. For example, the word BURN can be defined as a synonym for the CUT command. These statements begin with the symbol %SYN, followed by a major or minor word, and a user-definable alias name.

The following sections in this chapter explain how to construct phrases to write CVNC grammar statements.

Figure 2-2 Parts of Grammar File Statements



Phrases

Phrases contain one or more major and minor words, symbols, and special characters.

Major and Minor Words

Major words represent a main purpose or action (a CVNC command, macro, or pass-through statement). Minor words are modifiers to commands and macros, or arguments to pass-through statements. For example, in the system-supplied CVNC-M2 language, the major word, POCKET, is modified by several minor words.

Table 2-2 Major and Minor Words

POCKET	REV	REVFIN	QUERY
	MAXSTEP	APPROX	CANTED
	SIDE	RTOL	ISLAND

Please note: Pass-through statements can begin with a major word used in a CVNC command, but the statements and command syntax cannot be identical.

Symbols

All symbols begin with a percent sign (%). Symbols are used in phrases to specify

- The type of a statement
- Numeric or text data (expressions, constants, and variable names)
- Coordinate data (explicitly, by digitize, or by NCGROUP name)
- Entity selection (by tag name, by digitize, or by NCGROUP name)
- Choices or options
- Limits and repetition of data
- Online documentation files

The symbols, listed in the table below, are grouped with major words, minor words, and other symbols to create valid phrases. How to construct phrases with each of these symbols is explained in the rest of this chapter.

Please note: You can create macros and pass-through statements; you cannot re-create CVNC commands. Most statement examples provided in this chapter are of existing CVNC commands (already in the CVNC grammar). These examples provide models of grammar file statements, but they cannot be appended to the grammar, because duplicate commands are not allowed in a grammar.

Table 2-3 Symbols

Symbol	Purpose
%CHOOSE	Specifies mandatory options
%DOC	References an online documentation file
%ENDLOC	Specifies one or more x, y, z coordinate locations at the end of an entity
%ENT	Specifies single entity selection
%HIDDEN	Hides the word, metasymbol, bracketed construct, or macro it follows. Hides an entire statement when it follows %STMT and %REPEAT
%INVALID	Specifies invalid entity types
%LIMIT	Specifies a limit for entering data
%MACRO	Identifies a CVMAC macro with a fixed number of arguments
%MAV	Sets system variables
%MCHOOSE	Specifies options that may be chosen
%MGIVE	Specifies an option that may be chosen
%NEXP	Specifies a numeric expression
%OR	Separates options
%ORGLOC	Specifies one or more x, y, z coordinate locations at the origin of an entity
%REPEAT	Specifies a repeating phrase
%STMT	Identifies a pass-through statement
%SYN	Defines a synonym for a major or minor word
%TEXP	Specifies a text expression
%VALID	Specifies valid entity types
%VMACRO	Identifies a CVMAC macro with a variable number of arguments
%VAR	Specifies a numeric or text variable
%XYZLOC	Specifies one or more x, y, z coordinate locations
%XYLOC	Specifies one or more x, y coordinate locations

Special Characters

You can define special characters, such as the equal sign (=), in a phrase, so that the character is required in a command’s syntax.

Inserting Comment and Continuation Lines

Insert comment and continuation lines anywhere in a grammar file.

Comment Lines

Comment lines inserted in a file are not read by the system when the file is compiled. Comments are intended to make the file easier to read. Comments may be text explaining what the grammar file is doing, or empty comments inserted to improve readability. Any characters following comment symbols are ignored when the grammar file is processed.

To specify a comment line, begin the line with a pound sign (#).

Continuation Lines

Lines in a statement exceeding one screen line (72 characters) are called continuation lines. You must use continuation lines so that the system knows the same line is continuing and a new line is not starting.

To specify a continuation line, end the line with the left angle bracket and pound sign symbol (<#). Follow this with the right angle bracket (>) as the first nonblank character in the continuation of the line.

You can also insert comment lines between the continuation lines to improve readability.

Please note: The limit of characters allowed in a single line is 72; the total limit of characters allowed in a statement is 3200. If a statement exceeds this limit, use metasymbols in place of groups of phrases. This decreases the number of continuation lines.

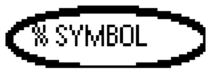
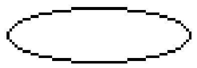
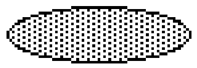
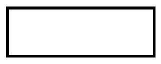
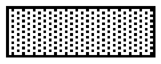
Continuation Lines

```
<#
************************************************************
********
<#
<# GRAMMAR FILE FOR MILLING
<#
<#
************************************************************
********
```

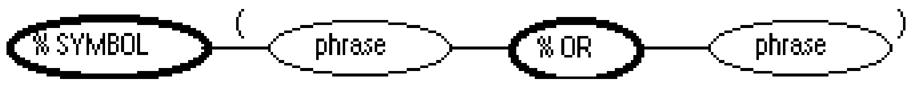
```
 %STMT CHGTOOL  %DOC 'CHGTOOLM2'
<#
>            %MGIVE( MANUAL )
<#
>            %MCHOOSE( CLW      %OR CCLW)
<#
>            $NUMNAM
<#
>              %MCHOOSE( DIAREG   %NEXP    %MGIVE ( LENREG
%NEXP) <#
>                 %OR  LENREG   %NEXP    %MGIVE ( DIAREG
%NEXP))  <#
>            %MGIVE( SFIG  %MGIVE ( %TEXP)) %MGIVE( NOOUTPUT
)
```

Format of Grammar File Phrases

The table below lists parts of the diagrams used to describe the format of grammar file statement phrases.

Part of Diagram	Definition
	A required symbol (preceded by a %)
	Requirement of one phrase
	Requirement of one or more phrases
	Optional entry of one phrase
	Optional entry of one or more phrases
( )	Required with parenthesis.

Example: The following format diagram shows the format for a %CHOOSE phrase:



Where

phrase is a minor word, symbol, or metasymbol that may be chosen. Choosing one option is mandatory.

%OR is a required symbol. It separates optional phrases. You may separate as many options as you wish.

Notice that the options are enclosed in parentheses.

An example of a %CHOOSE phrase is

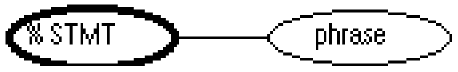
```
%CHOOSE ( %VAR %OR OFF %OR ON %OR LAST %OR NEXT)
```

In CVNC, this statement results in syntax where, if the relevant command is entered, you must then enter a variable or one of the minor words OFF, ON, LAST, or NEXT.

%STMT — Statement

%STMT is a statement identification symbol that specifies a pass-through statement. Begin all grammar statements for pass-through statements with %STMT.

Format



Where

phrase is one or more phrases defining the statement.

Example

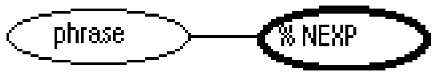
The following phrase (from the existing CVNC grammar) defines the syntax for the AUDIT command. The %DOC, %CHOOSE, and %OR symbols will be described later in this chapter.

```
%STMT AUDIT %DOC ‘AUDIT’ %CHOOSE ( ON %OR OFF )
```

%NEXP — Numeric Expression

Use the %NEXP (numeric expression) symbol in a statement where a numeric expression may be required.

Format



Example

The following phrase (from the existing CVNC grammar) specifies that a numeric expression must be entered after the SELECT LAY command.

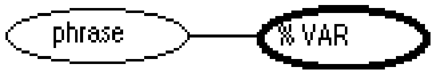
```
%STMT SELECT LAY %NEXP
```

%VAR — Variable

%VAR is used in a phrase requiring a numeric or text variable.

Use the %VAR symbol in a statement where a numeric or text variable may be required.

Format



Examples

All of these examples are from the existing CVNC grammar.

```
%STMT CALL %VAR
```

This phrase specifies that a variable name must be entered after the CALL command.

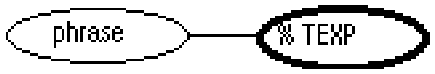
```
%STMT LET %VAR = %CHOOSE( %NEXP %OR %TEXP )
```

This phrase sets %VAR with a numeric expression or a text expression. The equal sign (=) is required in the syntax.

%TEXP — Text Expression

Use the %TEXP (text expression) symbol in a statement where a text expression may be required.

Format



Example

The following phrase (from the existing CVNC grammar) specifies that the major word (in this case, a pass-through statement) PPRINT must be followed by a text string (enclosed in quotes).

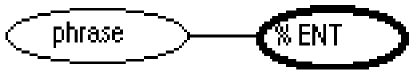
```
%STMT PPRINT %TEXP
```

%ENT — Entity

Use the %ENT (entity) symbol in a statement where an entity may be required.

%ENT requires that only one entity is entered. A %REPEAT or %LIMIT phrase must be used when more than one entity can be entered.

Format



Example

The following phrase specifies that an entity is required after the minor word TANTO.

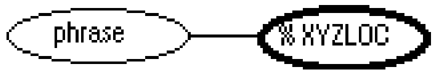
```
%STMT PROCEED TANTO %ENT
```

%XYZLOC — X, Y, Z Location

Use the %XYZLOC (x, y, z location) symbol in a statement where an x, y, and z coordinate may be required. It defines the equivalent of specifying LOC in getdata during a CADDS operation.

%XYZLOC requires that only one coordinate is entered. A %REPEAT or %LIMIT phrase must be used when more than one coordinate can be entered. These symbols are described later in this chapter.

Format



Example

The following phrase specifies that an x, y, z location is required after the minor word TO.

```
%STMT PROCEED TO %XYZLOC
```

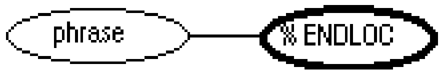
Please note: While in CVNC, you are in Model mode. When coordinate data is on the screen, it is in respect to the active coordinate system; when coordinate data is passed to macros, it is in model coordinates. How macros handle coordinate data is described in Chapter 4, “Customizing with CVMAC Macros”.

%ENDLOC — End Location

Use the %ENDLOC (end location) symbol in a statement where a coordinate may be required that must be obtained from the end of an entity. This symbol invokes the equivalent of specifying END in getdata during normal CADDS operation.

%ENDLOC requires that only one location is selected. A %REPEAT or %LIMIT phrase must be used when more than one coordinate can be entered. These symbols are described later in this chapter.

Format



Example

The following statement specifies that an x, y, z location at the end of an entity is required.

```
%STMT PROCEED TO %ENDLOC
```

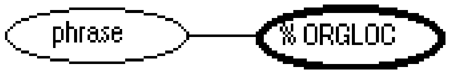
Please note: While in CVNC, you are in Model mode. When coordinate data is on the screen, it is in respect to the active coordinate system; when coordinate data is passed to macros, it is in model coordinates. How macros handle coordinate data is described in Chapter 4, “Customizing with CVMAC Macros”.

%ORGLOC — Origin Location

Use the %ORGLOC (origin location) symbol in a statement where a coordinate obtained from the origin of an entity may be required. CVNC prompts you to select an ORG when entering getdata. This symbol invokes the equivalent of specifying ORG in getdata during normal CADDS operation.

%ORGLOC requires that only one location is selected. Either a %REPEAT or %LIMIT phrase must be used when more than one coordinate can be entered. These symbols are described later in this chapter.

Format



Example

The following statement specifies that an x, y, z location at the origin of an entity is required.

```
%STMT PROCEED TO %ORGLOC
```

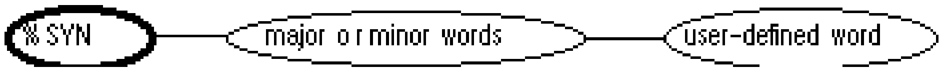
Please note: While in CVNC, you are in Model mode. When coordinate data is on the screen, it is in respect to the active coordinate system; when coordinate data is passed to macros, it is in model coordinates. How macros handle coordinate data is described in Chapter 2, “Using CVNC Grammar Files.”

%SYN — Synonym

You can define synonyms for any CVNC major word or minor word with the %SYN symbol. For example, synonyms allow you to tailor the CVNC language to a foreign language or specific class of machine tools.

Synonym words are not added to the user interface display; they do not appear in popup and pulldown menus.

Format



Where

major or minor word is the existing major or minor word for which you are specifying a synonym.

user-defined word provides a synonym for the major or minor word.

Example

When the following statement becomes appended to the CVNC grammar, CVNC interprets the word BURN to be the same as the CVNC major word CUT.

```
%SYN CUT BURN
```

%DOC — Documentation

Use the %DOC symbol to reference an online documentation text file. CVNC accesses the file when you enter a keyword followed by an exclamation point (that is, POCKET!) at the `NC:>` prompt. You can reference online documentation after any keyword in your grammar file.

CVNC online documentation files are in the following directory:

```
/usr/apl/cadds/data/cam/ncproc/doc/_bcd
```

The system looks in this directory for the file listed after the %DOC symbol.

Customizing Online Documentation

You can customize online documentation with new files for your macros and pass-through statements. You can also edit the existing documentation files to incorporate site-specific information and procedures.

Adding a New File

To add a new online documentation file, follow these steps:

1. Create a text file documenting your macro or pass-through statement in the directory:

```
/usr/apl/cadds/data/cam/ncproc/doc/_bcd.
```

2. Use the %DOC symbol followed by the name of the text file (in quotes) in the grammar file statement for your macro or pass-through statement.

Editing an Existing File

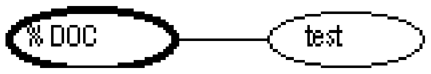
To edit an existing online documentation file, follow these steps:

3. Change to the CVNC documentation directory:

```
%cd /usr/apl/cadds/data/cam/ncproc/doc/_bcd
```

4. Edit the appropriate text file with any editor.

Format



Where

text is the name of the text file. It must be enclosed in single quotes. The text file must be located in:

```
/usr/apl/cadds/data/cam/ncproc/doc/_bcd
```

Example

The following phrase references the online documentation file for the PROFILE command.

```
%STMT PROFILE %DOC 'PROFILE'
```

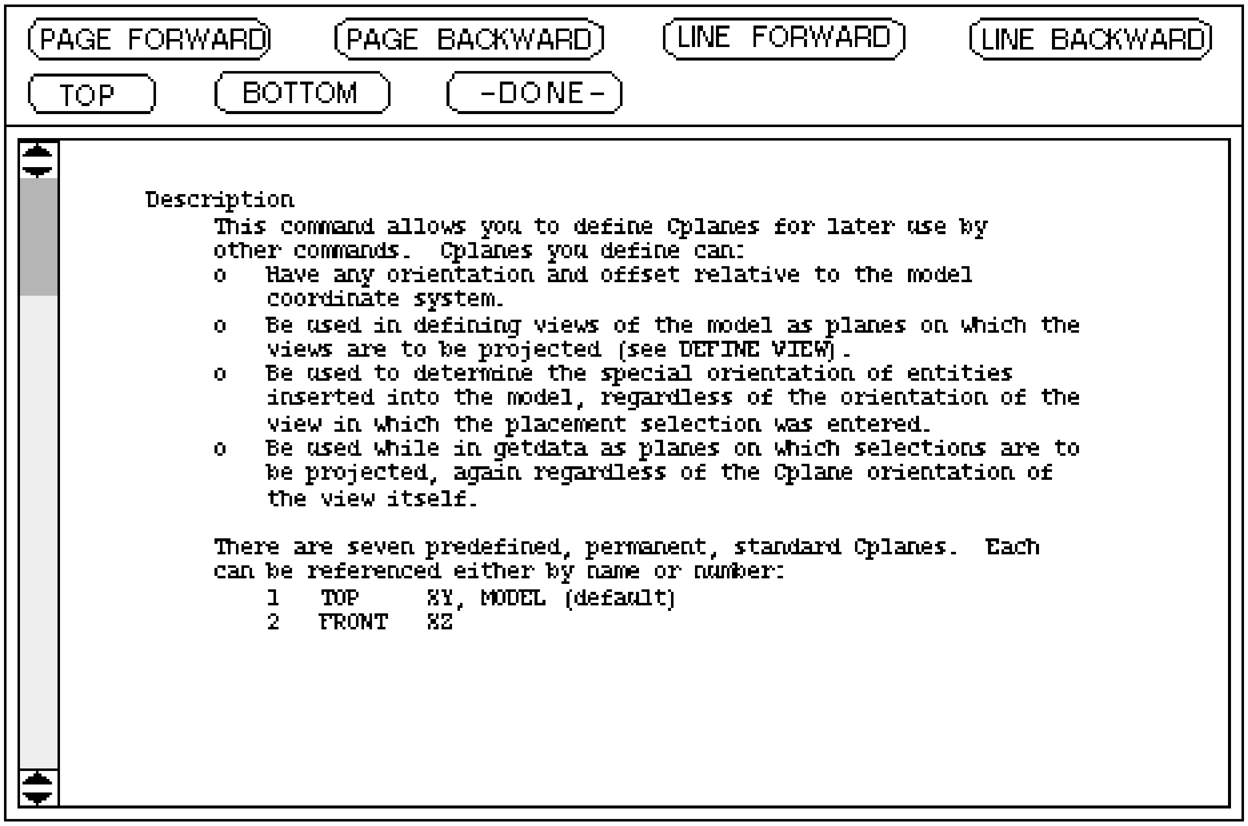
When a user enters

```
NC:>PROFILE!
```

An online documentation window will open (see the figure 1-3) displaying the contents of the text file:

```
/usr/apl/cadds/data/cam/ncproc/doc/_bcd/profile
```

Figure 2-3 Online Documentation Window



```
     Description
          This command allows you to define Cplanes for later use by
          other commands.  Cplanes you define can:
          o   Have any orientation and offset relative to the model
              coordinate system.
          o   Be used in defining views of the model as planes on which the
              views are to be projected (see DEFINE VIEW).
          o   Be used to determine the special orientation of entities
              inserted into the model, regardless of the orientation of the
              view in which the placement selection was entered.
          o   Be used while in getdata as planes on which selections are to
              be projected, again regardless of the Cplane orientation of
              the view itself.

          There are seven predefined, permanent, standard Cplanes.  Each
          can be referenced either by name or number:
              1   TOP     XY, MODEL (default)
              2   FRONT   XZ
```

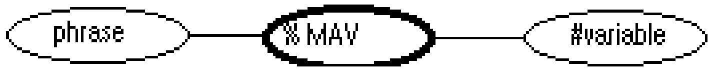
%MAV — System Variables

Use the %MAV symbol to set the value of a system variable from within a pass-through statement. A set of system variables, called user-definable system variables (listed in the following table), are reserved for your definition. For more information on setting user-definable system variables, see Chapter 3, “Customizing with Pass-through Statements.”

Table 2-4 User-Definable System Variables

Variable Name	Type
#TUSR01- #TUSR10	text
#NUSR01- #NUSR60	real
#CUSR01- #CUSR20	coordinate

Format



Where

phrase is made up of one or more minor words, symbols, special characters, and metasymbols.

#variable is one of the system variables reserved for user-definition.

Example

The following statement allows you to set two system variables.

```
%STMT MACHINE %TEXP %MAV #TUSR06 %MGIVE    <#
>   ( %NEXP %MAV     #NUSR40 )
```

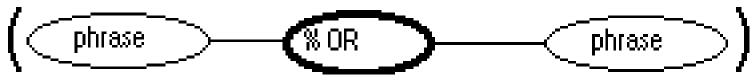
The following command places ‘VERTMILL’ in #TUSR06, and ‘9’ in #NUSR40

```
NC:> MACHINE ’VERTMILL’ 9
```

%OR

Use the %OR symbol with %CHOOSE or %MCHOOSE to separate options, such as the option of one minor word in a list of several.

Format



Where

phrase is made up of one or more minor words, symbols, special characters, and metasymbols.

Notice that the options are enclosed in parentheses.

Example

The following phrase contains three minor words, ON, PAST, and TO. They are separated by %OR symbols.

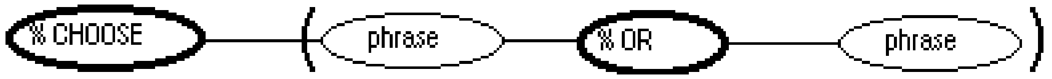
```
(ON %OR PAST %OR TO)
```

%CHOOSE

Use the %CHOOSE symbol where a choice of one option from a selection of many is mandatory.

You may nest %CHOOSE phrases inside each other or inside %MCHOOSE, %MGIVE, or %REPEAT phrases.

Format



Where

phrase is a minor word, symbol, or metasymbol. Choosing one option is mandatory.

%OR is a required symbol. It separates optional phrases. You may separate as many options as you wish.

Notice that the options are enclosed in parentheses.

Example

The following statement allows you to choose one option of several after inputting the major word STATUS. Note that the %OR symbol separates the choices.

```
%STMT STATUS %CHOOSE (%VAR %OR OFF %OR ON %OR<#
>             LAST %OR NEXT)
```

In the following phrase, you must select one of the options ON, PAST, or TO.

```
%CHOOSE (ON %OR PAST %OR TO)
```

%MCHOOSE — May Choose

Use the %MCHOOSE symbol in the same way you use the %CHOOSE symbol. %CHOOSE specifies a mandatory selection, whereas %MCHOOSE specifies an optional selection. Use %MCHOOSE when a selection is not necessary for a command's validity, or when you have programmed a default.

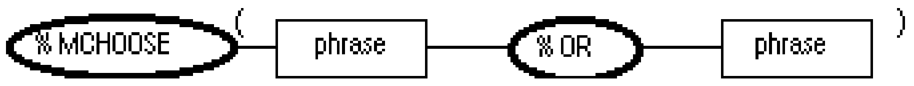
You may nest %MCHOOSE phrases inside each other or inside %MGIVE or %REPEAT phrases. %MCHOOSE is not valid in a %MACRO statement.

Please note: Although it is possible to nest %MCHOOSE phrases, use %CHOOSE constructs instead of %MCHOOSE wherever possible. Excessive nesting of %MCHOOSE phrases leads to duplicate displays of options in the pop-up windows and excessive grammar file size. In the example.

```
%MCHOOSE (LEFT %OR %MCHOOSE (RIGHT %OR ON) ) NEXT
```

You can replace %MCHOOSE with %CHOOSE, allowing the same options but resulting in a more compact grammar without duplication.

Format



Where

phrase is a minor word, symbol, or metasymbol, that may be chosen.

%OR is a required symbol. It separates optional phrases. You may separate as many options as you wish.

Notice that the options are enclosed in parentheses.

Example

In the following statement (from the existing CVNC grammar), the %MCHOOSE phrase specifies that you can enter XYPLANE, YZPLANE, or ZXPLANE after entering LEFT, RIGHT, OFF, or ON:

```
%STMT DIACOMP %MGIVE (%NEXP)                        <#
>   %CHOOSE  (LEFT %OR RIGHT %OR OFF %OR ON)        <#
>   %MCHOOSE (XYPLANE %OR YZPLANE %OR ZXPLANE)
```

%MGIVE — May Give

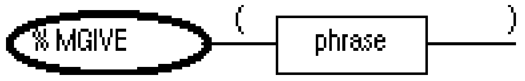
Use the %MGIVE symbol to specify a single option. This symbol allows a single option to be included or omitted from a command when it is executed.

You can nest %MGIVE phrases inside each other or inside %CHOOSE, %MCHOOSE, or %REPEAT phrases. %MGIVE is not valid in a %MACRO statement.

Please note: Although it is possible to nest %MGIVE phrases, use %CHOOSE constructs instead of %MGIVE wherever possible. Excessive nesting of %MGIVE phrases (and mixing of %MGIVE and %MCHOOSE phrases) leads to duplicate displays of options in the pop-up windows and excessive grammar file size.

The general form of a %MGIVE phrase is as follows:

Format



Where

phrase is a minor word, symbol, or metasymbol that may optionally be input. This phrase must be enclosed in parentheses.

Example

In the following statement (from the existing CVNC grammar), you may input a variable after the major word CMFIL:

```
%STMT CMFIL %DOC 'CMFIL' %MGIVE ( %VAR )
```

In the following statement (from the existing CVNC grammar), you may input the modifier CPL followed by a text expression:

```
%STMT CPL %TEXP %CHOOSE( $CPLCH  %OR %XYZLOC )<#
>   %MGIVE ( CPL %TEXP )
```

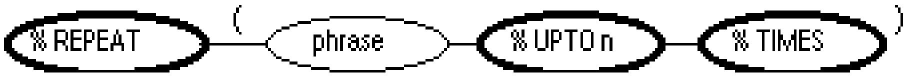
%REPEAT

Use the %REPEAT symbol when a phrase can be repeated up to a specified number of times. You can nest %REPEAT phrases inside each other or inside %CHOOSE, %MCHOOSE, or %MGIVE phrases.

%REPEAT is not valid in a %MACRO or %VMACRO statement.

The general form of a %REPEAT phrase is as follows:

Format



Where

phrase is one or more phrases.

%UPTO n %TIMES specifies the maximum number of times the phrase may be repeated.

Note the parentheses.

Example

The following statement (from the existing CVNC grammar) specifies that you may input the minor word MIRROR, and you must input XAXIS, YAXIS, or ZAXIS. Both the %MGIVE and %CHOOSE phrases may be repeated a maximum of three times.

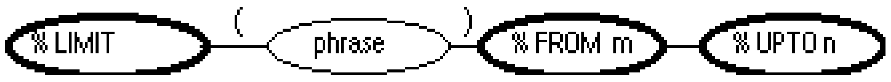
```
%STMT SETAX %REPEAT ( %MGIVE( MIRROR)<#
>      %CHOOSE ( XAXIS<#
>      %OR  YAXIS <#
>      %OR  ZAXIS ))<#
>       %UPTO 3 %TIMES
```

%LIMIT

Use the %LIMIT symbol to increase the quantity of data input, by specifying an upper and lower limit of data required. When used alone, phrases indicating coordinate or entity selection allow for only one item of data at a time.

This symbol is only valid with coordinate locations or entity selections (when data is digitized or explicitly entered).

Format



Where

phrase is a %XYZLOC, %XYLOC, %ENDLOC, or %ORGLOC symbol. This phrase requires parentheses.

%FROM m is the lower limit.

%UPTO n is the upper limit.

Example

The following statement shows how you use the %LIMIT construct to ensure that two coordinate locations are always supplied to define a vector in DIRVEC. You would only have to go into getdata once to enter both locations.

```
%STMT PROCEED %CHOOSE ( ON %OR PAST %OR TO ) %ENT <#
>     %MCHOOSE ( DIRLIN %ENT<#
>     %OR            <#
>     DIRVEC %LIMIT (%XYZLOC) %FROM 2 %UPTO 2<#
>     %OR            <#
>     DIRLOC %XYZLOC<#
>     %OR            <#
>      DIREND %ENDLOC)
```

If the statement asked for two %XYZLOCs (that is, DIRVEC %XYZLOC %XYZLOC) the same amount of data would be required, but you would have to go into getdata twice.

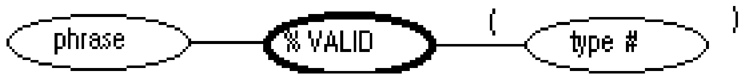
The following statement specifies that 1 to 100 entities may be entered after the major word GROUP:

```
%STMT GROUP %LIMIT ( %ENT ) %FROM 1 %UPTO 100
```

%VALID

Use the %VALID symbol to specify valid entities for a phrase. Specify entities with the corresponding data type numbers. For example, a line entity is a type 3. Only entity types listed after %VALID can be used at that point in the command syntax. A complete list of type numbers is provided in Appendix A, “Entity Types List”

Format



Where

phrase is one or more phrases for which you are specifying valid entities.

type # is a list of one or more valid entity type numbers. Separate each number with a comma. Enclose the list in parentheses.

Example

This statement uses two %VALID phrases.

VALID (2, 5) specifies that after inputting OPTIM ENT, only points and arcs are valid entities.

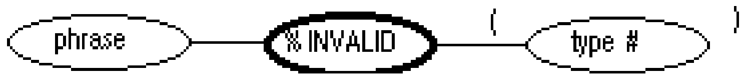
VALID (2,3,5,6,8,9,12,70) specifies that after inputting ENT, only the following types of entities are valid: points, lines, arcs, conics, B-splines, Cpoles, Nsplines, and strings.

```
%STMT NCGROUP %VAR<#
>  %CHOOSE ( OPTIM<#
>  %CHOOSE (ENT %LIMIT (%ENT) %FROM 1 %UPTO 100<#
>  %VALID (2,5)  <#
>  %OR  LOC %LIMIT (%XYZLOC) %FROM 1 %UPTO 50)<#
>  %OR  ENT  %LIMIT (%ENT)    %FROM 1 %UPTO 100<#
>  %VALID (2,3,5,6,8,9,12,70)<#
>  %OR  LOC %LIMIT (%XYZLOC) %FROM 1 %UPTO 50 )
```

%INVALID

Use the %INVALID symbol to specify invalid entities for a phrase. Specify entities with the corresponding data type numbers. For example, a line entity is a type 3. All entity types, except those listed after %INVALID, can be used at that point in the command syntax. A complete list of type numbers is provided in Appendix A, “Entity Types List.”

Format



Where

phrase is one or more phrases for which you are specifying invalid entities.

type # is a list of one or more invalid entity type numbers. Separate each number with a comma. Enclose the list in parentheses.

Example

```
%STMT ENTGROUP %VAR<#
> %CHOOSE ( OPTIM<#
> %CHOOSE ( ENT %LIMIT (%ENT) %FROM 1 %UPTO 100<#
> %INVALID (5)   <#
```

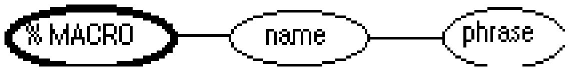
The %INVALID (5) specifies that after inputting OPTIM ENT, all entities except arcs are valid.

%MACRO

%MACRO statements link CVNC language and CVMAC macros. Phrases input as part of a macro statement are passed as arguments to the corresponding CVMAC macro. The phrase/argument correspondence is fixed for %MACRO statements (as opposed to %VMACRO statements).

If you use the %MACRO construct, you must ensure a one-to-one correspondence between the number of arguments you enter and the number of arguments in the PROC statement of the macro. You cannot use the following symbols in your %MACRO statement: %MGIVE, %MCHOOSE, %REPEAT and %MAV. Chapter 4, “Customizing with CVMAC Macros,” provides more information on using CVMAC macros.

Format



Where

name is a major word naming the CVMAC macro.

phrase is one or more phrases being passed as arguments to the corresponding macro.

Example

The %MACRO statement is

```
%MACRO INSARC RAD %NEXP AGO %NEXP AEND %NEXP %XYZLOC
```

The PROC statement in the macro is

```
PROC INSARC (&RAD, RAD, &AGOA, AGOA, &AENDA, AENDA, @D1)
```

There is a one-to-one correspondence between the number of arguments you enter and the number of arguments in the PROC statement.

```
%MACRO Used with %LIMIT and %VALID
```

In the following %MACRO statement, you must digitize two entities after entering the major word INSURF. Only the following types of entities are valid: lines, arcs, conics, B-splines, Cpoles, and Nsplines.

```
%MACRO INSURF %LIMIT (%ENT) %FROM 2 %UPTO 2<#
>  %VALID (3,5,6,8,9,12)
```

The PROC statement in the macro is

```
PROC INSURF($ents)
```

The variable is declared in the macro as follows:

```
DECLARE ENTITY $ents(2)
```

The complete INSURF macro is shown in Chapter 4, “Customizing with CVMAC Macros.”

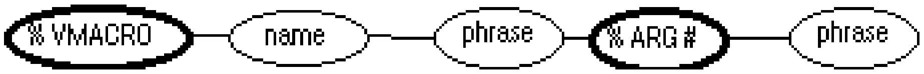
%VMACRO (Variable Argument Macro)

Macro statement definitions provide the link between CVNC and a user-written CVMAC macro. Phrases input as part of a macro statement are passed as arguments to the corresponding CVMAC macro.

Unlike %MACRO, %VMACRO allows you to define a macro that can take a variable number of arguments. The %ARG symbol specifies the sequential number of the argument corresponding to the order of arguments in the PROC statement of the macro.

You cannot use the %REPEAT or %MAV symbol in a %VMACRO statement.

Format



Where

name is the name of the CVMAC macro.

phrase is one or more phrases being passed as arguments to the corresponding macro.

%ARG n specifies the sequence number of the argument.

%ARG Rules

When using %ARG, the following rules apply:

1. Argument sequence numbers may appear in any order.
2. If you use the same argument more than once in a syntax statement, the last one given is used.
3. If the %ARG sequence number is greater than the number of arguments in the PROC statement, CVMAC treats it as an error and does not execute the macro.
4. Structure the macro to allow for the possibility that not all the arguments will be given values by the user input. The CVMAC construct APLINF/READ indicates this condition by returning a zero for any argument not input. The macro should then assign an appropriate default.

Argument Types

Use %ARG for any of the following with a %VMACRO statement:

- Any CVNC reserved (major or minor) word.
- Numeric data. Numeric data is passed to the macro as double precision real values.
- Text data entered. Text data is passed to the macro as a CVMAC text string.
- Coordinate data entered. Coordinate data is passed to the macro as a CVMAC vector array.
- Entity selection data to be entered. Entity selection data is passed to the macro as a CVMAC entity array.

Chapter 4, “Customizing with CVMAC Macros,” provides more information on CVMAC macros.

Example

In the following %VMACRO statement, only the variables specified by %ARG 1, %ARG 2, %ARG 3, and %ARG 4 are included in the PROC statement.

```
%VMACRO INSARC RAD %NEXP %ARG 1 AGO<#
>   %NEXP %ARG 2 AEND %NEXP %ARG 3<#
>   %XYZLOC %ARG 4
```

The PROC statement in the macro is as follows:

```
PROC INSARC (RAD, AGOA, AENDA, @D1)
```

The variables are declared in the macro as follows:

```
DECLARE DOUBLE RAD, AGOA, AENDA
DECLARE LOCATION @D1
%VMACRO Used with %MGIVE and %CHOOSE
```

In the following %VMACRO statement, there need not be a one-to-one correspondence between the number of arguments entered and the number of arguments in the PROC statement of the macro.

```
%VMACRO REAM DIAM %NEXP %ARG 1<#
> %MGIVE (FITTYPE %TEXP %ARG 2)<#
> %LIMIT (%XYZLOC) %FROM 1 %UPTO 300 %ARG 3<#
> %MGIVE (ENDLOC %LIMIT (%XYZLOC)<#
> %FROM 1 %UPTO 300 %ARG 4<#
> %CHOOSE (NOTOOL %ARG 5 %OR TOOLNUM %NEXP %ARG 6)
```

The PROC statement in the macro is as follows:

```
PROC REAM(%diam,&fit,@locs,@elocs,&notool,%tlnum)
```

The variables are declared in the macro as follows:

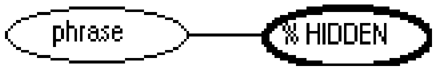
```
DECLARE LOCATION @locs(300),@elocs(300)
```

The complete REAM macro is shown in Chapter 4, “Customizing with CVMAC Macros.”

%HIDDEN

Use the %HIDDEN symbol to hide a word, metasymbol, bracketed construct, or macro within a statement. You can also use it to hide an entire statement following the %STMT and %REPEAT symbols.

Format



Example

All these examples are from the existing CVNC grammar.

```
%STMT AUDIT %DOC 'AUDIT' %CHOOSE ( ON %HIDDEN %OR OFF)
%STMT AUDIT %DOC 'AUDIT' %CHOOSE ( ON %OR OFF %HIDDEN )
```

In the first example, the phrase hides the word ON in the statement. In the second example, the phrase hides both choices in the %CHOOSE.

```
%STMT FLY $FOO %HIDDEN
```

This phrase hides the metasymbol within the statement.

```
%VMACRO DRILMAC PROMPT %NEXP %ARG 1 %HIDDEN
```

This phrase hides the numerical expression node in the grammar.

```
%STMT %HIDDEN AUDIT %DOC 'AUDIT' %CHOOSE ( ON %OR OFF )
```

This phrase hides the entire grammar of this statement at all levels of entry.

```
%MACRO %HIDDEN INSLIN %DOC 'INSLIN' %LIMIT (%XYZLOC) %FROM 2
%UPTO 2
%MACRO INSLIN %HIDDEN %DOC 'INSLIN' %LIMIT (%XYZLOC) %FROM 2
%UPTO 2
```

In the first example, the phrase hides the macro at all levels of entry. In the second example, the phrase hides the INSLIN macro name from the pop-up and displays the rest of the macro entered subsequently.

Special Characters

You may want a special character to be part of a command, macro, or pass-through statement’s syntax. Examples of special characters are an equal sign (=), a comma (,), and square brackets ([ ]).

The following characters are reserved and cannot be used:

?	!	”	’	$	#	%

Please note: If you want to use parentheses as part of a command, they must be enclosed in quotes, as in the following example:

```
%STMT MAJORWORD “(” MINORWORD “)”
```

Example

The following statement requires that the equal sign be input after %VAR:

```
%STMT LET %DOC ‘LET’ %VAR = %CHOOSE (%NEXP %OR %TEXP)
```

Using Metasymbols

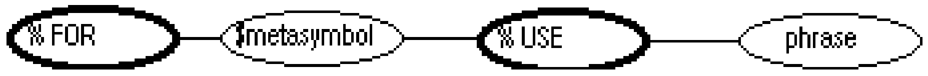
A metasymbol is a name given to a group of phrases. Metasymbols allow you to replace long phrases and sequences of phrases with just one word. Metasymbols begin with a dollar sign ($).

You can use a metasymbol anywhere a phrase is valid. You can also use the names of predefined metasymbols in metasymbol definitions. Note that CVNC does not allow recursion (when a metasymbol references itself).

Metasymbol definitions must begin with a %FOR symbol. Metasymbols are not valid in %MACRO and %VMACRO statements.

To define a metasymbol, use the following format:

Format



Where

%FOR is the symbol that begins a metasymbol definition.

$metasymbol is the name of the metasymbol, following a dollar sign ($).

%USE is the symbol that precedes the phrase(s) the metasymbol will replace.

phrase is one or more phrases that define the metasymbol.

Example

The following metasymbols are defined for the STOCK command.

```
%FOR $DRV  %USE  DRIVE  %NEXP
%FOR $CHK  %USE  CHECK  %NEXP
%FOR $EXPR %USE %NEXP $LENT2AX %REPEAT<#
>  ( %NEXP $LENT2AX ) %UPTO    10 %TIMES
%FOR $LENT2AX %USE %LIMIT(%ENT ) %FROM 1 %UPTO 200<#
>  %VALID (3,5,6,8,9,12,70)
%FOR $RSET %USE RESET %MCHOOSE ( $LENT2AX %OR ALL )
```

The statement for STOCK looks like this:

```
%STMT STOCK %DOC 'STOCK'  %CHOOSE<#
>    ( $DRV %MGIVE($CHK) %OR $CHK %MGIVE ($DRV)<#
>      %OR $EXPR %OR $RSET %OR ARCFLOAT %OR ARCFIX )
```

Restrictions on Grammar File Contents

Restrictions on grammar file contents are as follows:

- Do not use phrases for entity identification or coordinate locations as alternatives to each other.

  For example, you may not enter

```
%STMT CUT %CHOOSE ( %XYZLOC %OR %ENT )
```

  If you do, whenever you enter a colon (:) after the word CUT, CVNC will not know which option is required. The getdata feature would then always prompt for locations and accept the first option entered. To resolve this problem, use a minor word, as follows:

```
%STMT CUT %CHOOSE ( %XYZLOC %OR ENT %ENT )
```

- Do not use %VAR as an alternative to %TEXP or %NEXP. For example, the following statement is invalid:

```
%STMT NCGROUP %CHOOSE ( %VAR %OR %TEXP )
```

- Create synonyms for major or minor words *only.*
- Do not use numeric expressions and special characters, which are also operators, as alternatives in a %CHOOSE, or %MCHOOSE phrase. For example, the following statement is invalid:

```
%STMT CUT %CHOOSE ( %NEXP %OR + %OR -)
```

- The following characters are not allowed in a grammar file statement, unless they are part of a symbol (%) or a user-definable system variable (#):

?	!	”	’	$	#	%

- Do not add motion-generating macros to the rougher grammar file, as the macros will not interact with the material model. Other machine control commands are acceptable, however.
- Tabs are not allowed in a grammar file.
- Blank lines are not allowed in the grammar file.
- The limit of characters allowed in one line of a statement is 72; the total limit of characters allowed in a statement is 40,000.

Modifying the Grammar

This section explains how you can modify the grammar.

Methods of Modifying the Grammar

You can modify CVNC grammar by

- Adding new statements for user-written macros or pass-through statements
- Modifying existing statements for CVNC-supplied macros or pass-through statements, to suit your site's needs

New Statement

You can modify the CVNC grammar by appending statements for

- CVMAC macros that you have written
- Pass-through statements not already included in the files

To append to the grammar files, use the `ncgram append` command.

Existing Statements

You can modify CVNC grammar by appending modified statements for CVNC-supplied macros and CVNC-supplied pass-through statements. These statements are contained in the append text files: append-mac, append-post, and append-compact, which are located in grammar/_bcd.

To append to the grammar files, use the `ncgram append` command.

The procedure for editing pass-through statements is explained in Chapter 3, "Customizing with Pass-through Statements." The procedure for editing macros is explained in Chapter 4, "Customizing with CVMAC Macros."

Using the ncgram append Command

This section explains the use of the ncgram append command.

When to Use ncgram append

Use the operating system-level command ncgram append whenever you modify a grammar file to update the binary files used by CVNC. This command updates the binary files that CVNC uses during input of CVNC commands. You can append the following types of files using ncgram append:

- User-written text files containing grammar statements for CVMAC macros, pass-through statements, and metasymbols.
- Append text files (append-mac, append-pass, append-compact) containing grammar statements for CVNC-supplied macros, APT and CLFile pass-through statements, and COMPACT II pass-through statements.

Please note: To append text files containing macro grammar file statements (%MACRO and %VMACRO), you can use macutil instead of ncgram append. macutil appends an existing grammar, in addition to compiling and linking the macro.

Procedure

Before using ncgram append, make a copy of the grammar files to which you are going to append files. ncgram append overwrites the `_voc` and `_grm` files with your modified files. Always append files to a copy of the grammar, and keep the original grammar unchanged.

Please note: When using ncgram append, enter file names in CADDS format (that is, =dir.dir1.dir2.file1). For a list of CADDS and operating system file naming conventions, see Appendix B, “CADDS/UNIX File Name Character Conventions.”

To use ncgram append, enter

```
%ncgram append
```

ncgram append asks you to enter the following required information.

1. `Enter name of the directory containing text file:`

   This is the directory that contains the text file you wish to append. The file must be directly located under the directory `_bcd`. However, do not include the `_bcd` when you enter the directory name. For example, if the text file you are appending is located under my/grammar/_bcd, enter my.grammar.

2. `Enter name of text file:`

   This is the name of the text file that defines the syntax for a macro or pass-through statement.

3. `Enter name of directory containing grm & voc files:`

   This is the name of the directory where you copied an existing set of grammar files (`_grm` and `_voc` files) to which you will append the text grammar file. These grammar files are then replaced with files that contain the additions.

4. `Do you wish to check APT statements:`

   You can answer YES or NO (press RETURN if your answer is NO). If you are not entering APT/CLFile words, always reply NO.

   If you answer YES, ncgram append checks the words in any statement in the grammar file to see if they are valid APT/CLFile words. This is done by referencing the following files:

   ```
   /usr/apl/cadds/data/cam/aptwords/_bcd/major
   /usr/apl/cadds/data/cam/aptwords/_bcd/minor
   /usr/apl/cadds/data/numcon/apt/synonyms
   ```

   The synonyms file contains synonyms for some major and minor words. You can define a pass-through statement using these synonyms instead of the actual APT/CLFile words. (The synonyms file is not related to synonyms defined with the %SYN symbol.)

   If a word is not in any of these files, a warning message is displayed, but execution continues and the statements are entered into the grammar file.

   Appendix C, “Pass-through Statements and Macros,” lists valid APT/CLFile major and minor words and synonyms. If generating CLFile output and appending a statement with words not included in this list, you must edit the major or minor word to include these words and their corresponding integer values.

   If you answer NO, no check is made on the statements and they are always appended to the grammar.

5. `Enter name of check file:`

   If you enter a file name, a check file by that name is created and the expanded form of each command in the grammar will be written to it. If you do not require a check file, press RETURN at this point.

Please note: A check file expands all CVNC commands, therefore it is a very large file (approximately 30 megabytes). If a check file is required, make sure you have sufficient disk space before creating it.

Where ncgram append Creates the New Grammar

ncgram append appends the text file and creates new grammar files, overwriting the existing `_grm` and `_voc` files. Where ncgram append creates these files depends on how your create directory is set up in your CVPATH.

For example, if your CVPATH is

```
CVPATH
'/usr/apl/cadds/data/modtab:/usr/apl/cadds/data:/usr2/cadds
:/usr2/cadds/parts/=C:.:/usr/apl/cadds:/usr/apl/cadds/data/
dmenu:/usr/apl/cadds/data/vntab:/usr/apl/cadds/data/cam/ncm
ill'
```

your create directory is `/usr2/cadds/parts.` ncgram append creates the grammar files under the parts directory. See Chapter 1, “Overview of CVNC Customization,” for more information on the effect of the CVPATH on ncgram append.

Example

In the following example, the text file `macrostmts` is appended to a copy of a CVNC grammar. When ncgram append is completed, the appended grammar is stored under `/usr2/cadds/parts/my/grammar`.

```
%ncgram append
Enter name of the directory containing text file: my.grammar
Enter name of text file: macrostmts
Enter name of directory containing grm & voc files:
my.grammar
Do you wish to check APT statements: no
Enter name of check file:
Processing text file
Creation complete, beginning correctness check
Correctness check complete, beginning copy
Compilation successful, no errors occurred
The following files have been created
my/grammar/_voc
my/grammar/_grm
```

Accessing the Appended Grammar

To access the appended grammar, configure CVNC so that the default grammar is the appended grammar, or use the LANGUAGE command to specify that CVNC use the appended grammar for the current CVNC session only. This is described in the next section.

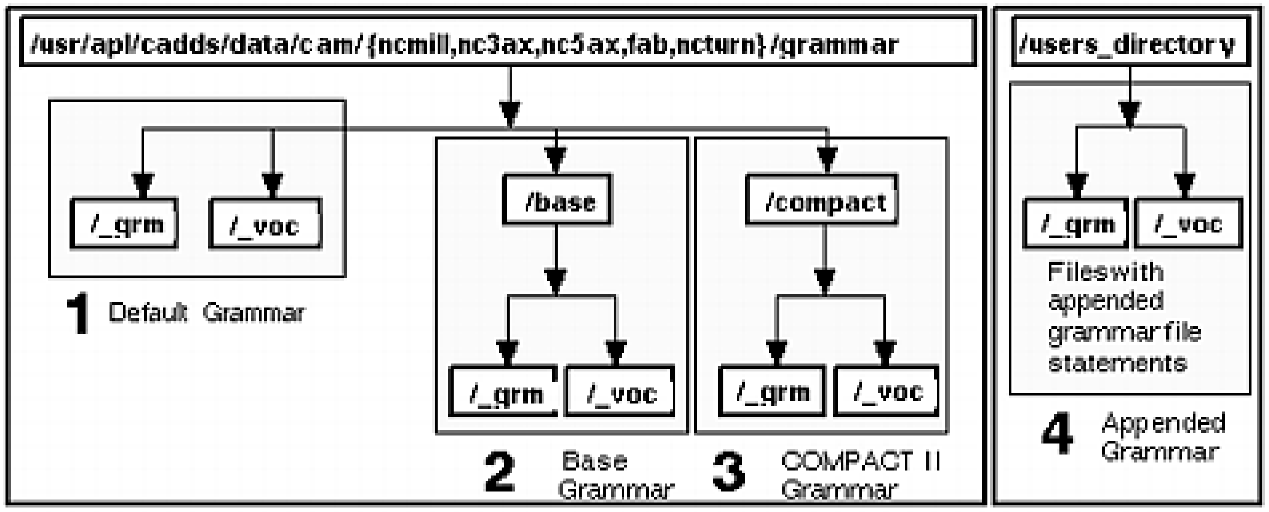
Using a Different Grammar

Unless you specify otherwise, CVNC uses the default grammar. The default grammar (item 1, in the figure below) contains CVNC commands for the product and commands to support a set of system-supplied CVMAC macros and pass-through statements for CLFile or APT output.

Other than the default grammar, you can use the grammar containing

- Base CVNC commands for the product, commands to support CVNC-provided CVMAC macros, and pass-through statements for CLFile and APT output. This is the default grammar.
- Base CVNC commands for the product, with no added macro or pass-through statements.
- CVNC commands for the product and CVNC-provided CVMAC macros and pass-through statements in the COMPACT II programming language.
- Grammar files to which you have appended statements for macros or pass-through statements. This grammar will be a copy of grammar 1, 2, or 3, plus statements for your own macros or pass-through statements.

The figure below shows where each grammar is located.



To use a different grammar, you can

- Configure CVNC so that it defaults to a different grammar

- Use the LANGUAGE command to specify a grammar for the current CVNC session only

Each method is described on the following pages.

Configuring CVNC to Change Grammars

There are two ways of configuring CVNC so that it defaults to a different grammar:

- Place a grammar earlier in the CVPATH
- Overwrite the original grammar

Placing a Grammar Earlier in the CVPATH

CVNC looks for the grammar located in the following directory path:

```
cam/{ncmill,ncturn,nc3ax,nc5ax,fab)/grammar
```

This path is found under `/usr/apl/cadds/data.` By creating the same path earlier in your CVPATH, CVNC will find that copy of the grammar first and use it, rather than the original grammar.

Example: In the following CVPATH, the /usr2/cadds directory is located before `/usr/apl/cadds/data/cam/ncmill` (that contains the default grammar). A copy of the base grammar is copied into the directory `/usr2/cadds/cam/ncmill/grammar`. Macros and pass-through statements can be appended to this grammar; CVNC uses it, rather than the default grammar.

```
CVPATH '/usr2/cadds:/usr/apl/cadds/data/modtab:
/usr/apl/cadds/data:/usr2/cadds/parts=C:.:/usr/apl/cadds:/u
sr/apl/cadds/data/dmenu:/usr/apl/cadds/data/vntab:/usr/apl/
cadds/data/cam/ncmill'
```

In this example, the steps are as follows:

1. Make a `cam/ncmill/grammar` directory path earlier in your CVPATH.

```
%cd /usr2/cadds
%mkdir cam cam/ncmill cam/ncmill/grammar
```

2. Copy the base grammar to this directory.

```
%cp /usr/apl/cadds/data/cam/ncmill/grammar/base/
{_grm,_voc} /usr2/cadds/cam/ncmill/grammar
```

Overwriting the Original Grammar

You can overwrite the grammar located in

```
/usr/apl/cadds/data/cam/{ncmill,ncturn,nc3ax,nc5ax,fab)
/grammar
```

with one of the other CVNC grammars or a grammar that you have appended.

Example

In the following examples, a `grammarsave` directory is created.

To reconfigure CVNC-M2 to use the grammar containing COMPACT II pass-through statements, follow this procedure:

1. Backup the original grammar files by copying them to a different directory. For example,

```
%cp /usr/apl/cadds/data/cam/ncmill/grammar/{_grm,_voc}
grammarsave
```

2. Overwrite the existing grammar files by copying the COMPACT II grammar files to them.

```
%cp /usr/apl/cadds/data/cam/ncmill/grammar/compact/
{_grm,_voc} /usr/apl/cadds/data/cam/ncmill/grammar
```

To reconfigure CVNC-M2 to use your own grammar (such as `/usr2/cadds/my/grammar)` to which macros or pass-through statements have been appended, follow this procedure:

1. Back up the original grammar files by copying them to a different directory.

```
%cp /usr/apl/cadds/data/cam/ncmill/grammar/{_grm,_voc}
grammarsave
```

2. Overwrite the existing grammar files by copying the appended grammar files to them.

```
%cp /usr2/cadds/my/grammar/{_grm,_voc}
/usr/apl/cadds/data/cam/ncmill/grammar
```

Using the LANGUAGE Command to Change Grammars

LANGUAGE specifies that CVNC use different grammar files for your current CVNC session and during subsequent reentries to the JCF created during that session.

Procedure

To use LANGUAGE, enter

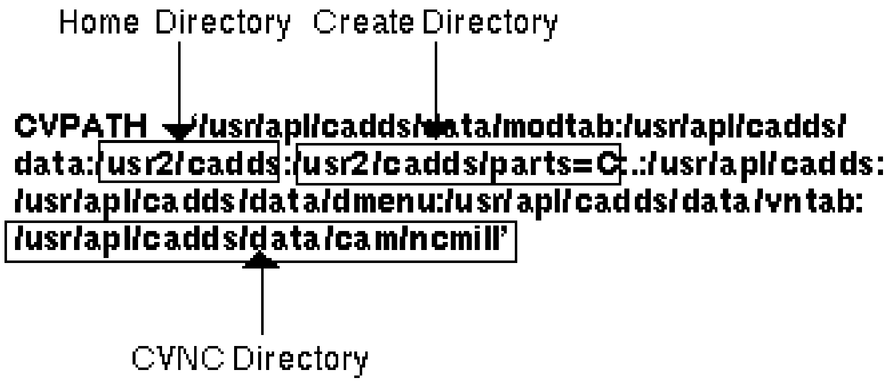
```
NC:>LANGUAGE directory
```

where directory is the directory containing the compiled grammar files to be used in the active CVNC session.

If directory resides in one of the CVPATHs specified in the `.caddsrc` file, you do not need to enter the full path name. If directory does not reside in your CVPATH, you must enter the full path name.

Example

In the following CVPATH, the user's home directory is /usr2/cadds. The create directory is /usr2/cadds/parts.



```
CVPATH '/usr/apl/cadds/data/modtab:/usr/apl/cadds/
data:/usr2/cadds:/usr2/cadds/parts=C:.:/usr/apl/cadds:
/usr/apl/cadds/data/dmenu:/usr/apl/cadds/data/vntab:
/usr/apl/cadds/data/cam/ncmill'
```

The CVNC directory in this CVPATH allows access to the compiled grammar files in the following directories:

`/usr/apl/cadds/data/cam/ncmill/grammar` (this is the default grammar)

```
/usr/apl/cadds/data/cam/ncmill/grammar/compact
/usr/apl/cadds/data/cam/ncmill/grammar/base
```

To specify the COMPACT II grammar instead of the default, enter

```
NC:>LANGUAGE ‘grammar.compact’
```

A full path name is not necessary since grammar.compact is in the CVPATH.

To specify files that are not in the CVPATH, enter the full path name.

```
NC:>LANGUAGE “=usr2.cadds.parts.my.grammar”
```

This command specifies appended grammar files stored under the `/usr2/cadds/parts/my/grammar.`

Remember to use CADDS file naming conventions when entering the directory (that is, use a period instead of a slash to separate directories). For a chart of basic CADDS file naming conventions, see Appendix B, “CADDS/UNIX File Name Character Conventions.”

Chapter 3

Customizing with Pass-through Statements

CVNC is equipped with predefined output pass-through statements, which are listed in the command reference for your application. This chapter explains how to customize CVNC by adding pass-through statements that CVNC does not provide.

- Pass-through Statements in Customization
- Adding Pass-through Statements
- Creating a Text File
- Modifying CVNC-supplied Pass-through Statements
- Editing Append Text Files
- User-definable System Variables
- Setting User-definable System Variables

Pass-through Statements in Customization

Output pass-through statements contain APT, CLFile, or COMPACT II words. They are entered like other CVNC commands, but they do not affect CVNC processing. For example, the following APT language pass-through statement specifies the machine tool for the postprocessor:

```
MACHIN "CINCIN.1"
```

This statement is passed directly to the output file in this format:

```
MACHIN/ CINCIN.1
```

It does not affect CVNC processing.

Constructing Pass-through Statements

Pass-through statements are constructed of phrases and metasymbols. Phrases, as discussed in Chapter 2, “Using CVNC Grammar Files,” contain major and minor words, symbols, special characters, and metasymbols.

Major and Minor Words

Pass-through statement major and minor words, supported by CVNC, are located in the following files:

```
/usr/apl/cadds/data/cam/aptwords/_bcd/major
```

```
/usr/apl/cadds/data/cam/aptwords/_bcd/minor
```

A third file, `/usr/apl/cadds/data/numcon/apt/synonyms`, contains synonyms for some major and minor words. You can define a pass-through statement using these synonyms instead of the actual APT words. For example, the following two grammar file statements are equivalent:

```
%STMT SELCTL %NEXP
%STMT SEL %NEXP
```

If you are generating CLFile output and write a statement with words not included in one of these files, you must edit the appropriate file so that it is included. See Appendix C, “Pass-through Statements and Macros,” for a list of all the APT/CLFile words and synonyms supported by CVNC.

TEXTOUT Command

The CVNC command TEXTOUT also passes data directly through to the output file. It passes a text string directly to an output APT or COMPACT II source file. For CLFiles, it passes the data to formulate a binary CLFile record. For more information on TEXTOUT, see the *CVNC System User and Menu Reference Guide.*

Methods of Customizing with Pass-through Statements

You can customize with pass-through statements by

- Adding new pass-through statements
- Modifying CVNC-supplied pass-through statements

Pass-through statements can also define system variables from a set of variables that CVNC reserves for you. These are called user-definable system variables.

Updating the User Interface

When you append a grammar file statement to the grammar for a new pass-through statement, the user interface updates to include the new statement. When help is used, the major and minor words and data prompts appear in the popup menus and status window.

Adding Pass-through Statements

You can add additional pass-through statements to CVNC, using one of the following methods:

- Creating a new text file
- Modifying the append text file containing CVNC-supplied pass-through statements

Method 2 allows you to include all pass-through statements in one file, but it requires that you append to the base grammar (which contains only CVNC commands). The other grammars already include the pass-through statements defined in the append text files; a grammar cannot include duplicate statements. If you modify the append text file and then append it to the base grammar, you must also append the append text file for macros. It is not included in the base grammar, either.

Creating a New Text File

You can add pass-through statements by creating your own text file and appending this file to the grammar with the `ncgram append` command. You can create one text file and use it like the CVNC-supplied append text file. Whenever you wish to add a new pass-through statement, edit that file.

Modifying the Append Text File

You can add pass-through statements by adding grammar file statements to the append-post or append-compact text file. To do this, follow the procedure in the module, Modifying CVNC-supplied Pass-through Statements. When editing the text file, refer to Chapter 2, CVNC Grammar Files, to construct a valid grammar file statement.

Appending a Text File Again

You cannot append a file to the same grammar more than once. If you modify a text file, you must append that file to a new copy of the original base grammar.

Creating a Text File

To create a text file containing pass-through statements that CVNC does not provide, follow this procedure:

1. With any editor, create a text file containing %STMT grammar file statements for the pass-through statements you want to add to the grammar. Follow the grammar construction rules in Chapter 2, "Using CVNC Grammar Files." This text file must be located under an `_bcd` directory.

   Refer to Chapter 2, "Using CVNC Grammar Files," for instruction on how to construct valid grammar file statements.

Please note: If you are generating CLFile output, the major and minor words contained in the file you create must be listed in the CVNC APT/CLFile major and minor word files, or the synonym file. See Appendix C, "Pass-through Statements and Macros," for a list of all the APT/CLFile words and synonyms supported by CVNC.

You can edit the appropriate file to include additional words and their corresponding integer codes.

2. Make a directory to store copies of the grammar files. For example,

```
% mkdir my/grammar
```

3. Make a copy of the grammar in the directory you just created. For example, to append a new pass-through statement to the default CVNC-M2 grammar, enter

```
% cp /usr/apl/cadds/data/cam/ncmill/grammar/{_voc, _grm}
my/grammar
```

Using `ncgram append`, you must append the following text files to a copy of the grammar:

- A text file, containing pass-through statements not supplied by CVNC, that you created or edited
- An append text file (append-post or append-compact) that you modified

See Chapter 2, "Using CVNC Grammar Files," for a step-by-step explanation of the ncgram append procedure.

Modifying CVNC-supplied Pass-through Statements

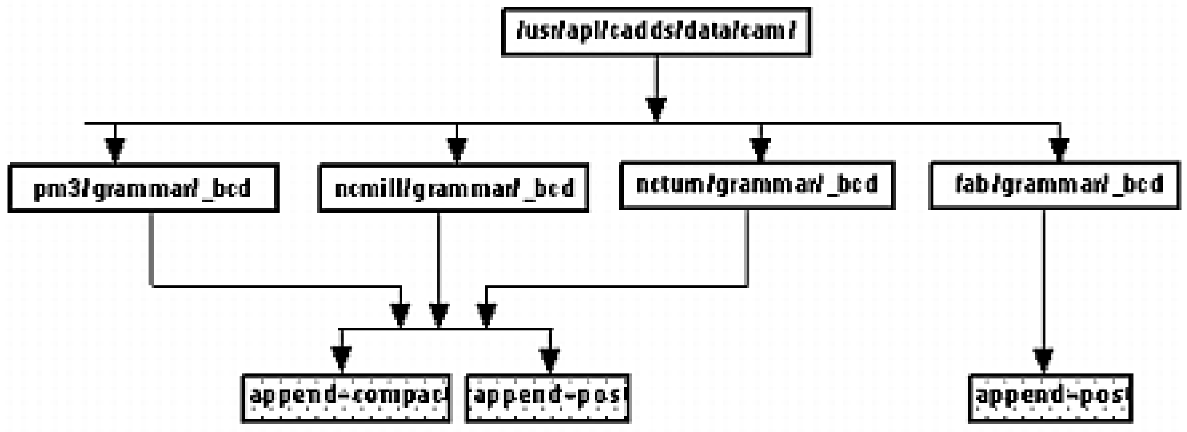
To modify CVNC-supplied pass-through statements, edit the append-post or the append-compact text files. These files contain grammar file statements for pass-throughs.

The append-post file contains APT and CLFile statements. The append-compact file contains COMPACT II statements.

There is an append-post file for CVNC-M2, CVNC-T2, and CVNC-P2. There is an append-compact file for CVNC-M2 and CVNC-T2 only. CVNC-M5 and CVNC-M3 use the CVNC-M2 pass-through statements. The following chart shows where append text files are located on the system; Appendix C, “Pass-through Statements and Macros,” shows their contents.

Please note: You can also modify the append text files to include grammar file statements for your own pass-through statements.

Figure 3-1 Append File Directories



Editing Append Text Files

To edit the append-post or append-compact file, follow these steps:

1. Make a directory to store copies of the append files and the grammar files. The append files must be in an `_bcd` directory. For example,

```
%mkdir my/grammar my/grammar/_bcd
```

2. Copy the `append-post` or `append-compact` file to your `_bcd` directory.

   For example, to modify an APT/CLFile pass-through statement, enter

```
%cp /usr/apl/cadds/data/cam/ncmill/grammar/_bcd/append-post
my/grammar/_bcd
```

3. Make a copy, in your grammar directory, of the Base binary files.

```
%cp/usr/apl/cadds/data/cam/ncmill/grammar/base/{_voc,_grm}
my/grammar
```

Please note: You must use the base grammar files when appending append text files. You cannot use the default grammar or the COMPACT II grammar since the append text files have already been appended. ncgram append will not allow you to append the same grammar statements more than once.

In your `my/grammar` directory, you should now have the following directories and files:

```
_bcd
_bcd/append-post
_voc
_grm
```

4. Edit the `append-post` file with any editor.

```
%vi my/grammar/_bcd/append-post
```

If you are generating CLFile output and add a major or minor pass-through statement word, verify that word is in CVNC APT/CLFile major and minor word files (see Appendix C, “Pass-through Statements and Macros”). If a word is not in one of these files, edit the major or minor word file to include it.

After editing the append text file, append it to the grammar, using `ncgram append`**.** See Chapter 2, “Using CVNC Grammar Files,”Chapter 2, for a step-by-step explanation of the `ncgram append` procedure.

User-definable System Variables

CVNC reserves a set of system variables that you can set (see the table below) in a pass-through statement. These variables, like the variables CVNC defines, can be numeric, text, and coordinate.

For more information on CVNC-supplied system variables, see the *CVNC System User Guide and Menu Reference.* For a list of system variables available in CVNC, see Appendix D in that book.

Table 3-1 User-definable System Variables

Variable Name	Type
#TUSR01-#TUSR10	Text
#NUSR01-#NUSR60	Real
#CUSR01-#CUSR20	Coordinate

Displaying System Variables

You can display the current values of system variables in the text window using the SHOW or PRINT command. The PRINT command provides only numeric and text values. PRINT becomes a permanent record in the JCF and serves as a check point throughout your work session. SHOW provides numeric, text, and coordinate values. It does not become a permanent record in your JCF.

System variables can also be displayed in the status display window. You can define the format of the status display with a status definition file. You can also turn different status displays on or off within a session. For more information on status definition files, see the *CVNC System User Guide and Menu Reference.*

Setting User-definable System Variables

To set user-definable system variables, append a pass-through statement to the grammar using ncgram append (see Chapter 2). The grammar statement for a pass-through statement must begin with %STMT, and use the %MAV symbol to set the value of the user-definable system variable.

The user-definable system variable is set when the pass-through statement is executed in CVNC.

Example: If you append the following statements to the grammar,

```
%STMT PRINTING %TEXP %MAV #TUSR01
%STMT DWELL %NEXP %MAV #NUSR01
%STMT ORIGIN %XYZLOC %MAV #CUSR01
%STMT MACHINE %TEXP %MAV #TUSR06 %MGIVE    <#>  ( %NEXP %MAV
#NUSR40 )
```

you can enter data in CVNC to correspond to the arguments within these statements.

The following command places ‘START OF OPERATION 2’ into #TUSR01:

```
NC:>PRINTING ‘START OF OPERATION 2’
```

The following command places 1 into #NUSR01:

```
NC:>DWELL 1
```

The following command places the coordinate 0, 2, 5 into #CUSR01:

```
NC:>ORIGIN X 0 Y 2 Z 5;
```

The following command places ‘VERTMILL’ in #TUSR06, and ’9’ in #NUSR40:

```
NC:>MACHINE ‘VERTMILL’ 9
```

To examine user-definable system variables once they are set, use the SHOW command.

```
NC:> NC:>DWELL 1
NC:>SHOW #NUSR01
NC:>Numeric Value:  1
```

Please note: You can reference user-definable system variables from within a macro using APLSYSR. This procedure is described in Chapter 4, “Customizing with CVMAC Macros.”

Chapter 4

Customizing with CVMAC Macros

CVNC macros are procedures written in the high-level language CVMAC. This chapter explains how to create macros to customize your CVNC system.

- Using CVMAC Macros
- Installing Macros
- CVMAC Interface with CVNC
- APLSYSR Statement
- APLINF Statement
- Creating Macros
- Macro Libraries
- Executing CVNC Commands within Macros
- Declaring Variables
- Mapping Coordinates
- Reprocessing READ and DIGI Statement
- Compiling and Linking Macros
- Compiling Macros
- Linking Macros
- Using the macutil Command
- Appending Macro Syntax to the Grammar
- Constructing Grammar File Statements for Macros
- Constructing %MACRO Grammar File Statements
- Constructing %VMACRO Grammar File Statements
- Specifying Your Macro Library
- Check Macros

- Creating a Check Macro
- Specifying Your Check Macro Library
- Using the checkutil Command
- Output Macros
- Creating an Output Macro
- Specifying Your Output Macro Library
- Point Macros
- Creating a Point Macro
- Generating Output with Point Macros
- Specifying Your Point Macro Library
- Configuring CVNC Macro Libraries
- Placing a Library Earlier in the CVPATH
- Overwriting the Original Library
- Modifying CVNC-supplied Command Macros
- Editing the Macro Text File
- Editing the append-mac File

Using CVMAC Macros

You can create macros to customize your CVNC system.

CVMAC Macros

CVNC macros are procedures written in the high-level language CVMAC. Macros contain logic to build intelligence into the system. CVMAC macros provide

- Programming logic for NC operations that can include branching and looping functions. For example, a macro may perform a set of moves at a specified depth, and then step down several times to repeat these moves until the required depth is reached.
- An interface to CADDS that extracts data from entities, creating geometric entities.

User-written Macros

CVNC supports four types of user-written macros.

- Command macros
- Check macros
- Output macros
- Point macros (for CVNC-M2, CVNC-M3, and CVNC-M5 only)

Command Macros

Command macros, like those provided with CVNC, are invoked at the CVNC prompt (`NC:>`). These macros can, for example,

- Read or write from files.
- Read or write from the CADDS data base.
- Perform algebraic calculations.
- Read the values of CVNC system variables and JCF variables.
- Issue CVNC commands. When a macro is run in CVNC to issue a sequence of CVNC commands, these commands are collected and automatically processed after the macro has completed its execution.

- Execute CVNC commands immediately as encountered during macro execution.
- Call another CVMAC procedure from within a CVNC macro (using the CVMAC CALL P command).

CVNC supplies a set of command macros already installed and ready to use. These macros provide various capabilities valuable for part programming with CVNC. In addition to this, the CVMAC and grammar statement text files that comprise these macros can be examined to provide examples of CVMAC macro programming and CVNC/CVMAC macro construction. Appendix C, "Pass-through Statements and Macros," shows the CVMAC text files for some of these macros.

It is also possible to customize CVNC-supplied macros for your specific needs.

Check Macros

Check macros are a set of procedures that is executed after processing an associated command. One macro can be implemented for each CVNC command. Each check macro sets a result flag as pass or fail. Check macros execute almost transparently during the CVNC session; they display error messages and terminate commands.

Output Macros

Output macros customize CVNC output generation, by sending pass-through statements directly to output. Output macros are activated with the OUTLIB ON/OFF [SYS] command in CVNC.

Point Macros

Point macros are accessed at predefined points during command execution. They can be written for the AREAMILL [M2], SURFCUT3 [M3], SURFCUT4 [M5 Four Axis], and SURFCUT5 [M5] commands. By issuing pass-through statements (or TEXTOUT statements) to the output, they allow you to add machine control statements within the tool paths created by AREAMILL or SURFCUT5.

Installing Macros

The following is a checklist for the steps necessary to install user-written macros and CVNC-supplied macros that you have modified. Later in this chapter, these steps are explained in detail with examples.

Steps for All User-written Macros

To install all user-written macros:

1. Create a source file in the CVMAC language using any editor.
2. Compile the macro using the cvmcomp command.
3. Create a link file using any editor. This link file lists a set of macros to be linked into a single library. It also provides a name for the library.
4. Link the macro to the library with the cvmlink command. This library contains the set of macros listed in the link file.

Steps for Command Macros

To install command macros, perform steps 1-4 and then

5. Create a text file containing the command syntax to be added to the language using any editor.
6. Create your own grammar file by copying the CVNC `_grm` and `_voc` files, and appending your additions using `ncgram append`.
7. Use the LANGUAGE command to activate the appended version of the language for the current CVNC session, or configure CVNC to use your grammar.

Steps 5-7 are explained in Chapter 2, “Using CVNC Grammar Files.”

8. Use the CVNC MACLIB command to identify the library containing your macros.

Please note: To install modified CVNC-supplied command macros, perform steps 1-4. Rather than creating a text file in step 1, edit the existing text file for the macro.Steps 5-7 are necessary only if the modification affects the grammar file statement. For step 5, rather than creating a new file, edit the append-mac file containing the grammar file statement for the macro you edited.

Steps for Check Macros

To install check macros, perform steps 1-4 and then

9. Use the CHECKLIB command to identify the library containing your check macros.

Steps for Output Macros

To install output macros, perform steps 1-4. If the macro generates a pass-through statement not already included in the CVNC grammar, perform steps 10 through 12.

10. Create a text file containing the syntax of the pass-through statements to be added to the language using any editor.
11. Create your own grammar file by copying the CVNC `_grm` and `_voc` files, and appending your additions using `ncgram append`.
12. Use the LANGUAGE command to activate the appended version of the language for the current CVNC session, or configure CVNC to use your grammar.

Steps 10-12 are explained in Chapter 2, "Using CVNC Grammar Files."

13. Use the CVNC OUTLIB command to identify the library containing your macros.

Steps for Point Macros

To install point macros, perform steps 1-4. Give the macro you create the same name as the predefined point at which you want to invoke the macro. The macro names are already determined for each predefined point in the commands. After step 13,

14. Use the MACLIB command to identify the library containing your point macros.

Point macros are automatically accessed whenever the predefined point is reached.

Please note: Point macros exist for the AREAMILL, SURFCUT3, SURFCUT4, and SURFCUT5 commands only. For a listing of CVNC point macros for these commands, see the *CVNC Milling Command Reference*.

CVMAC Interface with CVNC

The CVMAC statements APLSYSR and APLINF allow CVMAC to exchange data with CVNC.

- APLSYSR (application system read) reads a CVNC system variable or local variable into a CVMAC variable.
- APLINF (application information) statements provide the amount of data being passed from or to CVNC. They can also set CVNC variables from within macros.

APLSYSR Statement

APLSYSR reads the value of a CVNC system variable, a local variable, or user input.

Syntax of APLSYSR Statement

The general syntax of the APLSYSR statement is as follows:

APLSYSR/“CVNCVAR1”,MACVAR1,“CVNCVAR2”,MACVAR2...

Where

CVNCVAR1 and CVNCVAR2 may be either system variables (such as #FEDRAT, #ZWORK, #CURLOC) or local variables (as defined using the LET command [SYS]).

MACVAR1, MACVAR2 are variables defined within the CVMAC macro.

Defining and Reading Variables

Variables defined in CVNC (system variables or variables defined using LET or READ) are read using APLSYSR in any subsequent CVMAC macro. For example, if the following local variables are defined:

LET DOUG = 23.2
LET CLIFF = ‘HELLO’

then these values and the values of CVNC system variables could be read using APLSYSR in any CVMAC macro as follows:

- APLSYSR/“#FEDRAT”,FEED

  Loads the current cutting feed value into the CVMAC variable FEED (declared REAL).
- APLSYSR/“#NUSR01”,FEED

  Loads the value of the user-defined variable #NUSR01 into the CVMAC variable FEED (declared REAL). For more information on user-definable system variables, see Chapter 3, “Customizing with Pass-through Statements.”

- APLSYSR/“#CURLOC”,@XYZ

  Loads the current tool position into the CVMAC variable @XYZ (declared LOCATION or VECTOR).
- APLSYSR/“#TLNAME”,&TOOL

  Loads the current tool name into the CVMAC variable &TOOL (declared TEXT).
- APLSYSR/“#FEDRAT”,FEED,“#CURLOC”,@XYZ,“#TLNAME”,&TOOL

  Performs all three of the above functions.
- APLSYSR/“CLIFF”, &MACTXT, “DOUG”, MACVAL

  Loads the current value of CLIFF into the CVMAC variable &MACTXT, and the current value of DOUG into the CVMAC variable MACVAL (declared REAL).

You must define a CVNC local variable prior to its use in a CVMAC APLSYSR statement. Otherwise, CVNC issues an error message indicating that unknown text has been passed to the CVMAC application.

Data Types for System Variables

The data type for each system variable must match the data type of the corresponding CVMAC variable. If it does not, a runtime error occurs. The following table lists the correspondence necessary between data types:

Table 4-1 Correspondence between CVNC and CVMAC Variable Data Types

CVNC Variable Data Type	CVMAC Variable Data Type
Numeric in Macro ARG List	DECLARE DOUBLE
Numeric in APLSYSR	DECLARE REAL or DOUBLE
Text	DECLARE TEXT
Coordinate	DECLARE LOCATION or VECTOR

#INTRCT Variable

Normally, CVNC only reprocesses macro data that has changed, making macros quicker and more efficient. You can force a macro to always reprocess when it is reexecuted, or when you move forward (F) over it with the CVNC editor. To do this, include the #INTRCT variable in the APLSYSR statement.

#INTRCT forces reprocessing; it does not pass data to a CVMAC variable. #INTRCT must follow the same format as other variables in the APLSYSR statement. For example,

```
APLSYSR/“#INTRCT”, NUM,
“#FEDRAT”,FEED,“#CURLOC”,@XYZ
```

#INTRCT and the variable following it (NUM) will always equal 0.

APLINF Statement

APLINF/READ and APLINF/WRITE controls the amount of information coming from and passed to CVNC.

APLINF/READ, n, VAR Statement

The APLINF/READ statement provides information about input arguments; it provides the number of data items contained in a given argument.

Syntax

The syntax of an APLINF/READ statement is as follows:

```
APLINF/READ,n,VAR
```

Where

n is the argument sequence number in the PROC statement of the macro (first argument [1], second argument [2], third argument [3], etc.).

Please note: Up to 15 arguments are allowed in a PROC statement.

VAR is the variable that is set to the number of values in argument n.

Example

```
PROC MYMAC (@LISTA, @LISTB,@LISTC)      <#
APLINF/READ,1,COUNT
    .
    .
    .
RETURN
END
```

When APLINF executes, count is set to the number of elements in @LISTA, the first argument to MYMAC.

APLINF/WRITE, n, VAR Statement

The APLINF/WRITE statement provides information about output arguments. That is, it provides the number of data items to be returned in a given argument.

Syntax

The syntax of an APLINF/WRITE statement is as follows:

```
APLINF/WRITE,n,VAR
```

Where

n is the argument sequence number in the PROC statement of the macro (first argument [1], second argument [2], third argument [3], etc.).

Please note: Up to 15 arguments are allowed in a PROC statement.

VAR is the variable that is set to the number of values to be returned in argument n.

Example

```
PROC MYMAC (@LISTA, @LISTB, @LISTC)
          .
          .
          .
APLINF/WRITE, 2, COUNT
RETURN
END
```

When APLINF executes, it tells CVMAC to return COUNT data items in @LISTB, (the second argument to MYMAC).

Please note: You can only output to macro arguments that correspond to a CVNC variable.

The SWITCH macro (see the display on the next page) shows an APLINF/WRITE statement and an APLINF/READ statement. This macro moves the contents of a LOCATION NCGROUP from @GRPIN to @GRPOUT and reverses the order of the locations while doing so.

APLINF/READ reads into the CVMAC variable NIN the number of locations passed through the variable @GRPIN. APLINF/WRITE tells CVMAC that NIN locations are being returned through the second parameter of SWITCH: @GRPOUT.

The arguments of SWITCH require that you enter a variable name as the second argument. This may be a previously defined NCGROUP or previously unused variable name. In the JCF below, the NCGROUP contains 5 locations. Therefore, NIN is equal to 5.

```
NC:>LANGUAGE 'MY.GRAMFILE'
NC:>MACLIB 'USR.EXAMPLE.MACLIB'
NC:>NCGROUP DOUG LOC X1 Y1 Z0 X2 Y2 Z0 X3 Y3
      Z0 X4 Y4 Z0 X5 Y5 Z0;
NC:>SWITCH DOUG; JAMIE
```

Please note: When coordinate locations are passed from CVNC to CVMAC, they are translated to CADDS model space (the equivalent of the TOP Cplane). To use coordinates with respect to other Cplanes, use CVMAC mapping commands (as shown in the example later in this chapter). For more information on coordinate mapping, see the *CVMAC Language Reference*.

SWITCH Macro

```
PROC SWITCH (@GRPIN, @GRPOUT)
<# PURPOSE:
<#    Move the contents of a LOCATION NCGROUP from @GRPIN to
@GRPOUT
<#    and reverse the order of the locations while doing so.
<#
<#    The syntax allows digitizes, explicit coordinates, or
an NCGROUP
<#     name for input to @GRPIN, but insists on a variable
name (of an
<#     NCGROUP) being provided to match with @GRPOUT.
<#     This is necessitated by the fact that this macro will
<#     APLINF/WRITE to @GRPOUT, and variables are the only
syntax
<#     elements one can write to.
<#
<#     Declare input variables
      DECLARE LOCATION @GRPIN(100), @GRPOUT(100)
<#
<#     Declare internal variables
      DECLARE REAL I,NIN
<#  Get the number of locations in @GRPIN.
      APLINF/READ,1,NIN
      PRINT THERE ARE {NIN} LOCATIONS IN THIS GROUP:
<# Loop NIN times moving each entry to the 'opposite slot in
@GRPOUT
<#
        I=1
        WHILE (I.LE.NIN)
        @GRPOUT (NIN-I+1) = @GRPIN(I)
        I = I + 1
```

```
       ENDWHILE

<#
<#     Now, you tell the system to output data through the
second
<#    argument, and also note how much data is to be output.
<#
<#     In this case, you want to pass NIN data items
(coordinates) out in
<#     the second argument.
<#     Note that the APLINF/WRITE command does not provide
the data to be
<#     passed - that is done by virtue of the fact that you
have loaded
<#     @GRPOUT with data (coordinates), and argument #2 is
@GRPOUT.
<#
       APLINF/WRITE,2,NIN
<#
<#     Done
       RETURN
       END
```

Creating Macros

To use any of the following types of macros in CVNC, you must first create a macro text file:

- Command macros
- Check macros
- Output macros
- Point macros

To create a CVMAC macro, create a file in the CVMAC language using any editor. Macros are stored in macro libraries; macro libraries are activated as a single unit.

You can use most capabilities of CVMAC, such as mapping coordinates, in your macro. You can also execute CVNC commands from within a macro.

Please note: The CVMAC EXECDF command is not allowed in a CVNC macro.

For more detailed information on the CVMAC language, see the *CVMAC Language Reference.*

Macro Libraries

You can have several different macro libraries. Using the correct CVNC library command, you activate the library containing the macro you want to use. The type of macro decides which library command should be used.

- The library of command macros and point macros is specified in CVNC with the MACLIB command. For example,

```
NC:>MACLIB ‘MACRO.LIB’
```

  Up to four macro libraries are allowed in a single command.

- The library of check macros is specified in CVNC with the CHECKLIB command. For example,

```
NC:>CHECKLIB ‘CHECK.LIB’
```

  Only one check macro library is allowed in a single command.

- The library of output macros is specified in CVNC with the OUTLIB command. For example,

```
NC:>OUTLIB ‘OUT.LIB’
```

  Only one output macro library is allowed in a single command.

You can configure CVNC so that it defaults to the library of your choice, eliminating the need to specify the macro library in the JCF. This is described later in this chapter.

Executing CVNC Commands within Macros

You can execute CVNC commands within CVMAC macros. Within a CVMAC macro, you can insert CVNC commands by first entering either a single exclamation point (!) or a double exclamation point (!!).

Please note: CVNC editor commands (that is., F, B) are not allowed in macros.

CVNC command execution from within a macro provides the macro with greater flexibility, such as performing logical operations dependent upon the result of a specific CVNC command.

You can use the IN command in CVNC to examine the CVNC commands resulting from the macro's execution. All commands preceded by the double exclamation point will appear before those commands preceded by the single exclamation point.

Execution Order of Commands

! or !! followed by CVNC commands can appear anywhere within the macro between the PROC and END statements. However, CVMAC, directed by the logic of the macro, executes the CVNC or CADDS commands that follow these constructs in the order it encounters them in the macro.

Commands preceded by !! are executed immediately. Commands preceded by ! are saved in an internal command file and executed after the macro is finished.

Please note: CVMAC macros use ! and !! in CADDS and CVNC. However, the commands preceded by these constructs must be CVNC commands when you are in CVNC and CADDS commands when you are in CADDS.

Commands Following a Double Exclamation Point

When CVMAC encounters !! in a macro, it

- Substitutes all constructs within braces ({}) that occur after the double exclamation points.
- Halts execution of the macro.
- Passes to CVNC for validation of the command that follows.
- Terminates execution of the macro if CVNC encounters an error, and backs up all action of the macro up to that point.

!! further allows you to control macro execution with a CVNC command through the use of the APLSYSR statement, as in the following example:

Example

```
PROC IMMED ( A )
DECLARE DOUBLE A
<#
!! CUT XINC { A }
<#
APLSYSR/"#CURLOC",@LOC
<#
IF ( XCOMP ( @LOC ) .GT. 23 ) THEN
.
.
ELSE
```

Commands Following a Single Exclamation Point

When CVMAC encounters ! in a macro, it

- Substitutes all constructs within braces ({}) that occur after the !.
- Saves the resultant CVNC commands in an internal command file.
- Executes all the saved CVNC commands after the macro is finished.

If you had inserted a ! in the above example, then APLSYSR would have read in the #CURLOC when you called the macro because CVMAC does not execute the CVNC command CUT XINC until the macro is finished.

Example of command execution order is listed in the following table. This sequence, when entered in CVNC, sets FEED to 30.

Table 4-2 CVNC Command Syntax within Macros

Example	Explanation
!! FEED 10	CVMAC executes this command first.
! FEED 20	CVMAC executes this command third.
! FEED 30	CVMAC executes this command fourth.
!! FEED 40	CVMAC executes this command second.

Declaring Variables

When declaring variables, follow these rules:

1. All numeric variables in the argument list must be declared with a DECLARE DOUBLE statement (double precision). For example,

```
DECLARE DOUBLE A
```

declares the first argument in the following PROC statement:

```
PROC RICH (A)
```

and the grammar file statement

```
%MACRO RICH %NEXP
```

2. Numeric variables used in APLSYSR statements can be declared with either a DECLARE REAL (single precision) or DECLARE DOUBLE statement (double precision). For example,

```
DECLARE REAL THING
DECLARE DOUBLE THING
```

declare the variable following:

```
APLSYSR/"#FEDRAT", THING
APLSYSR/"#SPINSPD", THING2
```

3. Scalar variables do not need to be declared for text, entities, or locations.
4. Variable arrays must always be declared.

Mapping Coordinates

You will usually want to map coordinates passed to CVMAC from a macro command in CVNC, from model space to the active Cplane. When a macro is used in a Cplane other than TOP, the coordinates passed to the macro are relative to model space, even though they appear as the active Cplane coordinates on the command line.

Map only those coordinates passed from the CVNC command line or an APLSYSR statement. Do not map locations given interactively through a DIGI statement, or coordinates passed from an OBTAIN statement, which are in the active Cplane (see table below).

Table 4-3 Coordinate Variables Read Into CVMAC

Method	Construction Plane Variable Is Relative to...
From CVNC command (through macro argument list)	Model space
OBTAIN statement	Active Cplane
DIGI	Active Cplane
APLSYSR	Model space

Using Mapping Commands

If coordinate variables are passed from a CVNC command or an APLSYSR statement, use CVMAC mapping commands, as in the following example. For more information on mapping, see the *CVMAC Language Reference.*

Example

If the following command were entered in the front Cplane, the location variable @START would receive the coordinate X 5 Y -1 Z 3:

```
NC:>MAPMACRO START X 5 Y 3 Z 1;
```

The following CVMAC statements map the coordinates to the active Cplane:

```
GETPRM (ACS) $ACSP
MAP ‘TOP’, $ACSP, 1 @START, @STARTM
```

The variable @STARTM will contain the correct coordinate (X 5 Y 3 Z 1).

Reprocessing READ and DIGI Statement

If a macro includes a DIGI or READ statement, it requires user input. DIGI allows you to input locations, entities, and views. READ allows you to input a real number or a text string.

In the following cases, CVNC always stops processing a JCF to wait for READ or DIGI user input:

- When you reexecute the JCF using the EXEC modifier to the PROGRAM command.
- When you reexecute the JCF from an execute file using the CADDS EXECUTE FILE command.
- When the macro includes an APLSYSR statement with an #INTRCT variable.

When moving forward (F) over a previously executed macro that does not include #INTRCT in an APLSYSR statement, CVNC may stop processing the JCF to wait for READ or DIGI user input. This occurs if a parameter, such as a system variable, has changed since the last time the macro was executed.

Compiling and Linking Macros

Compile and link macros that you have created or modified. The following operating system level commands are used for compiling and linking macros:

- cvmcomp compiles CVMAC macros.
- cvmlink links CVMAC macros to a macro library. All macros linked to this library can be activated as one unit.
- macutil can be used instead of cvmcomp and cvmlink. It
    - Compiles a specific CVMAC macro or all macros under a specified directory.
    - Creates a link file, and performs the linking and loading of the required macros under the directory where the named macro exists.
    - Generates the appended grammar files you specified, under a directory below the macro directory. The default directory is gramdir. This directory provides the needed language environment for CVNC.
- checkutil is used for creating check macros. It
    - Creates CVMAC code for CVNC check macros
    - Compiles and links macros into one library
    - Allows you to add or modify macros in an existing library
    - Allows you to choose the directory where the check library will live
    - Allows you to name the check library
    - Provides online ‘help’ documentation for checkutil and creating check macros

Compiling and Linking Command and Output Macros

To compile and link a command macro (including CVNC-supplied macros that you have edited) or an output macro, use one of the following command sequences:

Sequence 1	Sequence 2
1.**cvmcomp**	1. **macutil**
2. **cvmlink**	This procedure also performs **ncgram append**.

Compiling and Linking Point Macros

To compile and link a point macro, use the following command sequence:

Sequence
1.**cvmcomp**
2. **cvmlink**

Compiling and Linking Check Macros

To compile and link a check macro, use one of the following command sequences:

Sequence 1	Sequence 2
1. **cvmcomp**	1. **checkutil**
2. **cvmlink**	This procedure also creates the macro for you.

Compiling and Linking Modified Macros

After you modify a macro, it must be recompiled. To compile and link an existing macro that you have modified, use one of the following command sequences:

Sequence 1	Sequence 2
1. **cvmcomp**	1. **macutil**
2. **cvmlink**	

Compiling Macros

Before a macro can be executed, it must be translated or compiled into a form the CVMAC Runtime System (RTS) can understand and execute. The cvmcomp command compiles a CVMAC source program into binary code that the CVMAC RTS interprets.

Procedure

To use cvmcomp, enter

```
% cvmcomp macroname
```

Where

macroname is the relative path name of the macro to be compiled.

Please note: Use CADDS conventions when entering the macro's path name (i.e., use a period instead of a slash to separate directories). For a chart of basic CADDS file naming conventions, see Appendix B, "CADDS/UNIX File Name Character Conventions."

Example

To compile a macro named insurf in the directory macro, enter

```
% cvmcomp macro.insurf
SETTING CADDS ENVIRONMENT
done.
COMPUTERVISION CVMAC COMPILER VERSION 1.00 7-10-89 13:28

UNDECLARED SYMBOLS

$ent

CODE MODULE SIZE  =  52
DATA MODULE SIZE  =  40
BUFFERED DATASIZE =  40
VIRTUAL  DATASIZE =  0

macro.insurf@cm FILED
macro.insurf@dm FILED
```

The UNDECLARED SYMBOLS message lists any undeclared variables. The UNSET SYMBOLS message is displayed after compilation (successful or not) if any arguments were not set in the program.

The compiler indicates the size (in 16-bit words) of the code module and data module. Also, it indicates whether all data fits into the CVMAC Runtime System page zero data buffer or whether some data has to be stored in the RTS virtual pool. For more information on the page zero data buffer and the virtual pool, see the *CVMAC Language Reference.*

Please note: cvmcomp creates the compiled files (macro/insurf.cm and macro/insurf.dm) in your create directory. This directory is determined by your CVPATH. For example, if your create directory is /usr2/cadds/parts, the following files are created:

```
/usr2/cadds/parts/macro/insurf.cm
/usr2/cadds/parts/macro/insurf.dm
```

For more information on where cvmcomp searches for and creates files, see Chapter 1, “Overview of CVNC Customization.”

Linking Macros

To link macros, create a CVMAC link file, and then link the macros with the cvmlink command.

Creating a CVMAC Link File

A link file is a text file containing the name of the macro or macros to be linked and the name of the macro library to which you are linking them. Create a link file before using cvmlink.

To create a link file, use any editor. The file must contain one MACROLIB statement, at least one LINKM statement, and one ENDLINK statement. The format is as follows:

```
MACROLIB libraryname
LINKM macroname
ENDLINK
```

Where

libraryname is the relative path name of the library for the macros. The default library is maclib. libraryname is used later by the CVNC MACLIB command to activate the link file.

macroname is the name of the compiled macro to be linked. You can list more than one macroname in a LINKM statement. For example,

```
LINKM macroname1, macroname2, macroname3
```

You can also write a separate LINKM statement for each macro you are linking. For example,

```
LINKM macroname1
LINKM macroname2
LINKM macroname3
```

Please note: Use CADDS conventions when entering the macro's path name (i.e., use a period instead of a slash to separate directories). For a chart of basic CADDS file naming conventions, see Appendix B, "CADDS/UNIX File Name Character Conventions."

Example

The link file (in this example, macro/linkfile) for the INSURF macro looks like this:

```
MACROLIB macro.maclib
LINKM macro.insurf
ENDLINK
```

MACROLIB indicates the beginning of creating a macro library in which to store INSURF. (You use MACROLIB instead of the CVMAC IMAGE command to create a CVNC macro library or check library.)

Linking the Macro with cvmlink

Use cvmlink to create a CVMAC library. cvmlink builds the macro file into a macro library. Macros stored in a library can be activated as a single unit.

To use cvmlink, enter

```
%cvmlink linkfilename
```

Where

linkfilename is the relative path name of the link file.

Example

You have written the link file (linkfile) for the INSURF macro under the directory macro. Enter,

```
%cvmlink macro.linkfile
SETTING CADDS ENVIRONMENT
done.

COMPUTERVISION CVMAC LINKING LOADER VERSION 1.00 7-10-89
13:31:22

CMEM IMAGE FILE macro.maclib@cm!
00000200 WORDS WRITTEN TO IMAGE
DMEM IMAGE FILE macro.maclib@dm!
00000100 WORDS WRITTEN TO IMAGE
```

When successfully executed, cvmlink creates the following three files:

- Code image file, which is stored as `libraryname.cm`. It contains binary code interpretable by the CVMAC RTS.
- Data image file, which is stored as `libraryname.dm.` It contains CVMAC data.
- Symbol table file, which is stored as *l*`ibraryname.sm.` It contains start addresses of individual macros.

Please note: cvmlink creates the compiled macro library files in your create directory, even if you override your CVPATH where the create directory is specified. For example, if your create directory is `/usr2/cadds/parts`, the following files will be created:

```
/usr2/cadds/parts/macro/maclib.sm
/usr2/cadds/parts/macro/maclib.dm
/usr2/cadds/parts/macro/maclib.cm
```

For more information on where cvmlink searches for and creates files, see Chapter 1, “Overview of CVNC Customization.”

Using the macutil Command

The `macutil` command automatically compiles and links macros and then appends them to the grammar.

`macutil` performs the functions of the `cvmcomp`, `cvmlink`, and `ncgram` append commands.

`macutil` enables you to

- Compile a specific CVMAC macro or all macros under a specified directory.
- Create a link file, and perform the linking and loading of the required macros under the directory where the named macro exists.
- Generate the appended grammar files you specified under a directory below the macro directory. This directory provides the needed language environment for CVNC.

Before Using macutil

Before using `macutil`:

1. Create the macro(s) you want to compile, link, and append to the grammar.
2. Create a text file containing the `%MACRO` or `%VMACRO` grammar file statements. This file may be a modified version of the append-mac file. Information on creating `%MACRO` or `%VMACRO` statements is provided later in this chapter, and in Chapter 2, “Using CVNC Grammar Files.”
3. Create a directory to store the appended grammar files that `macutil` generates. The default directory is gramdir.

Procedure

To use `macutil`, enter

```
%macutil
```

`macutil` asks you to enter the required information. Enter `h` for help at any `macutil` prompt.

Please note: Use operating system file conventions, not CADDS conventions as you do with `cvmcomp`, `cvmlink`, and `ncgram` append when using macutil.

1. `Type in the path name of the required macro or the directory which includes the required macro files`

   Enter the path name of the macro or the directory of more than one macro, that you want to compile, link, and append to the grammar.

2. `Type in the path name of the append/grammar text file`

   Enter the path name of the text file containing the `%MACRO` or `%VMACRO` grammar file statements. This file must be in an `_bcd` directory.

3. `Type in the directory name for source grammar files`

   Enter the name of the of the CVNC directory containing the grammar to which you are appending. For example, to append to the default NCMill grammar, enter `/usr/apl/cadds/data/cam/ncmill/grammar`. See Chapter 2, "Using CVNC Grammar Files," for more information on CVNC grammars.

4. `Type in the destination directory for language files`

   Enter the name of the directory you created to store the appended grammar files that macutil generates. The default directory is gramdir.

Example

The following example compiles and links the macro insurf to a grammar used for CVNC-M2. The path names of files and directories referenced in this example are as follows:

Current working directory	`/usr2/cadds`
Macro	`/usr2/cadds/macro/insurf`
Text file for grammar	`/usr2/cadds/my/grammar/_bcd/insurfstate`
CVNC source grammar files	`/usr/apl/cadds/data/cam/ncmill/grammar`
Directory of appended files	`/usr2/cadds/my/grammar`

```
% pwd
/usr2/cadds
% macutil
          **************************************
          *              MACUTIL               *
          *          CVMAC Macro Utility       *
          *                                    *
          *                                    *
          **************************************
```

```
 Enter 'h' for help at any prompt ...
 ------------------------------------
type in the pathname of the required macro or
the directory which includes the required macro file(s)  >*:
macro/insurf

Type in the path name of the append/grammar text file >*:
my/grammar/_bcd/insurfstate

Type in the directory name for source grammar files
>*:/usr/apl/cadds/data/cam/ncmill
/grammar
Type in the destination directory for language files >*:
my/grammar

 ----------------------- Compiling Macro(s)
---------------------------
compiling ** insurf **

UNDECLARED SYMBOLS
$ent
 CODE MODULE SIZE  =  52
 DATA MODULE SIZE  =  40
 BUFFERED DATASIZE =  40
 VIRTUAL  DATASIZE =  0
insurf@cm FILED
insurf@dm FILED
SETTING CADDS ENVIRONMENT
done.
  ----------------------- Linking & Loading Macro(s)
-------------------

CMEM IMAGE FILE maclib-insurf@cm!
00000200 WORDS WRITTEN TO IMAGE
DMEM IMAGE FILE maclib-insurf@dm!
00000100 WORDS WRITTEN TO IMAGE
SETTING CADDS ENVIRONMENT
done.

 ---------------- Generating Appended Grammar File
--------------------


Enter name of the directory containing text file:
Enter name of text file:
Enter name of directory containing grm & voc files:
Do you wish to check APT statements:
Enter name of check file:(ncgram append list is issued. It
displays te expanded command syntax.)
```

```
1!%VMACRO INSURF %LIMIT (%ENT) %FROM 2 %UPTO 2 <#
2!>      %VALID (3,5,6,8,9,12) %ARG 1
Processing text file
Creation complete, beginning correctness check
Correctness check complete, beginning copy
Compilation successful, no errors occurred
The following files have been created
gramdir/_voc
gramdir/_grm

In order to use this set of macro(s) inside CVNC,
                           the following commands should be
issued :
-----------------------------------------------------------
---------
               LANGUAGE    "=USR2.CADDS.MY.GRAMMAR"
               MACLIB      "=USR2.CADDS.MACRO.MACLIB-INSURF"
```

You can now access the macros in your appended grammar by typing the following:

```
NC:>LANGUAGE 'MY.GRAMMAR'
NC:>MACLIB 'MACRO.MACLIB-INSURF'
```

For more information on the LANGUAGE command, see Chapter 2, "Using CVNC Grammar Files."

Please note: The directories containing the appended grammar files and the macro library must be in your `CVPATH`. If they are not, enter a full path name with the LANGUAGE and MACLIB commands.

Appending Macro Syntax to the Grammar

For the following types of macros, you must append a statement to the grammar file that describes the syntax for a new macro.

- Command macros
- CVNC-supplied macros that have been edited so that the grammar file statement has changed

Please note: Output macros and reserve macros may contain pass-through statements that need to be appended to the grammar. For more information on pass-through statements, see Chapter 3, "Customizing with Pass-through Statements."

To append macros to the grammar,

1. Construct a valid grammar file statement, in a text file, that describes the command's syntax. Macro grammar file statements begin with a `%MACRO` or `%VMACRO` statement. They provide the link between CVNC and a user-written CVMAC macro. Phrases input as part of a macro statement are passed as arguments to the corresponding CVMAC macro.

   This text file must be located under an `_bcd` directory.
2. Use the `ncgram` append to append the text file to a copy of the grammar. This procedure is described in Chapter 2, "Using CVNC Grammar Files."

You can also use the `macutil` command to append the text file, if you want to compile, link, and append with one command.

Constructing Grammar File Statements for Macros

Macro grammar file statements define the syntax of the macro. Macro grammar file statements must begin with either `%MACRO` or `%VMACRO`. The main difference between `%MACRO` and `%VMACRO` is their flexibility.

With `%MACRO`, ensure a one-to-one correspondence between the number of arguments you enter and the number of arguments in the PROC statement of the macro.

Unlike `%MACRO`, `%VMACRO` allows you to define a macro that can take a variable number of arguments. The `%ARG` symbol specifies the sequential number of the argument corresponding to the order of arguments in the PROC statement of the macro.

Phrases input as part of a macro statement are passed as arguments to the corresponding CVMAC macro. The macro syntax defined in the grammar file statement must agree with the data types of the macro arguments. For example, a numeric argument must be declared with a DECLARE DOUBLE statement in the macro, and defined with a `%NEXP` symbol in the grammar.

The CVNC language provides four valid data types:

- Numeric
- Text
- Entity
- Locations

The table on the next page lists these data types and the corresponding argument and grammar file symbol.

Please note: Use the `%NEXP, %TEXP, %ENT, %XYLOC,` and `%XYZLOC` symbols for input to macros. They allow for a greater flexibility in the JCF input required. An exception to this is where a macro sets variables in the JCF; in this case, the statement definition must use a `%VAR`.

After creating a text file with the grammar file statement(s), append this file to the grammar using ncgram append. See Chapter 2, “Using CVNC Grammar Files,” for an explanation of this procedure.

Table 4-4 Macro Argument Data Types

Data Type	Macro Argument	Grammar File Symbol
Numeric	DECLARE DOUBLE	%NEXP, %VAR
Text	DECLARE TEXT	%TEXP, %VAR
Entity	DECLARE ENTITY	%ENT
Locations	DECLARE LOCATION	%XYLOC %XYZLOC %ENDLOC %ORGLOC

Constructing %MACRO Grammar File Statements

%MACRO statements link CVNC and a user-written CVMAC macro. The correspondence between a `%MACRO` statement and the macro's arguments must follow these rules:

- There must be a one-to-one correspondence between the number or arguments in the %MACRO statement and the PROC statement.

Please note: Up to 15 arguments are allowed in a PROC statement.

- Each keyword typed must be passed to the macro as TEXT. Each text string (in quotes) must also be passed as TEXT.
- Each numeric expression must be passed as a DOUBLE; each coordinate selection as a LOCATION; and each entity selection as an ENTITY (TEXT, DOUBLE, LOCATION, and ENTITY are variable declarations in CVMAC).
- Choices between %ORs must be the same in both number and data type to satisfy phrase/argument correspondence.

Please note: You cannot use metasymbols or the %MGIVE, %MCHOOSE, %MAV, and %REPEAT symbols in a %MACRO statement.

Example

The following grammar file statement describes the syntax for the INSURF macro. This macro, shown in the next figure, creates an NSURF (NURBS Surface) from two user-specified entities.

```
%MACRO INSURF %LIMIT (%ENT) %FROM 2 %UPTO 2 <#
>   %VALID (3,5,6,8,9,12)
```

This grammar file statement defines the following:

```
%LIMIT (%ENT) %FROM 2 %UPTO 2
```

Entities must be entered. A minimum of 2, and a maximum of 2 insures that two entities are entered.

```
%VALID (3,5,6,8,9,12)
```

Only lines, arcs, conics, B-splines, Cpoles, and Nsplines are valid entities.

The syntax for INSURF in CVNC, would look like this:

```
NC:> INSURF: Model ent d d
```

INSURF Macro

```
PROC INSURF($ents)
<#
<# INPUT ARGUMENTS :
<#
<#      $ents : The two entities that will be used to create
<#             the NSURF. The valid entity types that can be
<#              used to create a NSURF with this macro are:
<#              LINES, ARCS, CONICS, BSPLINES, CPOLES,
NSPLINES
<# GRAMMAR:
<#
<#      %MACRO INSURF %LIMIT (%ENT) %FROM 2 %UPTO 2 <#
<#      >%VALID (3,5,6,8,9,12)
<#
<# Example JCF:
<#
<#      LANGUAGE 'MYGRAMMAR'
<#      MACLIB 'MACRO.MACLIB'
<#      INSURF $L1 $L2;
<#
DECLARE ENTITY $ents(2)
DECLARE REAL narray(2),deg
<#
deg = 3
<#
        $ent = NSURF/deg,COUNT,2,$ents(1),narray
END
```

The following %MACRO grammar file statement describes the syntax for the ROUGH macro. This macro, shown in the figure below, generates multiple rough cuts to a predefined NCGROUP. %MACRO does not give you the flexibility of associating phrases in your command line to multiple argument statements in the macro. In the next module, ROUGH is defined with a %VMACRO statement where this flexibility does exist.

```
%MACRO ROUGH %CHOOSE (ON %OR TO)
<#     %TEXP
<#     %CHOOSE (OPEN %OR CLOSED)
<#     %CHOOSE (ROLL %OR NOROLL)
<#     ZSTEP %VAR
<#     ZEND %NEXP
<#   %VAR
```

The syntax for ROUGH in CVNC (not available in CVNC-M3) might look like this:

```
NC:> ROUGH TO 'SQUARE' CLOSED ROLL ZSTEP STEP ZEND -2
NUMCUTS
```

ROUGH Macro Using %MACRO

```
PROC ROUGH
(&ONTO,&BNDRY,&CLOSED,&ROLL,&ZSTEP,STEP,&ZEND,ZEND,NUMCUTS)
<#  PURPOSE:
<#      This macro generates multiple rough cuts to a
predefined
<#     NCGROUP.  The first cut begins at the location of the
tool
<#     (called CURLOC) when the macro is called, and is made
at
<#      the current cutting depth (ZWORK). Subsequent cuts
are made
<#     at equal z-decrements (not to exceed STEP) until ZEND
is
<#      reached
<#
<#     Between passes, a retract to the XY of CURLOC, but at
the
<#      retract Z, is inserted, followed by a plunge to the
next
<#      depth.
<#

<#     After the last pass, only the retract move is issued.
ZWORK
<#      will have been reset to ZEND by this time, and the
macro
<#      also outputs the calculated values for the actual
step size
<#      and the number of cuts.
<#
<#  COMMAND SYNTAX                   CORRESPONDING ARGUMENTS
<#
<#  %MACRO ROUGH %CHOOSE (ON %OR TO)<# ==> &ONTO
<#  > %TEXP <# ==> &BNDRY
<#  > %CHOOSE (OPEN %OR CLOSED) <# ==> &CLOSED
<#  > %CHOOSE (ROLL %OR NOROLL)<# ==> &ROLL
<#  > ZSTEP %VAR <# ==> &ZSTEP STEP
<#  > ZEND %NEXP                          <# ==> &ZEND ZEND
<#  > %VAR             <# ==> NUMCUTS
<#
<#  Description of arguments
<#  INPUT
<#     &ONTO  =    'ON' or 'TO' to describe desired profile
mode.
```

```
<#         Default = 'TO'.
<#         NOTE: CURLOC should be properly positioned with
<#         respect to the first boundary element to
correctly
<#         identify side-of-boundary for ON.
<#         &BNDRY  = Name of NCGROUP to be profiled.  It
must be
<#         comprised of at least 3 entities.  It is passed
into this macro
<#         as a text string (rather than variable or entity
<#         group) solely to facilitate the creation of the
<#         correct text for the profile commands to be
issued.
<#
<#      &CLOSED  'OPEN' or 'CLOSED' to describe the
boundary.
<#
<#      &ROLL      'ROLL' or 'NOROLL' to describe radial
motion
<#         around corners or not.
<#
>#      &ZSTEP      'ZSTEP' to identify next input as the
maximum step.
<#     STEP        The maximum step size — input a variable,
so
<#         that the actual (re)calculated step size may be
output.

>#     &ZEND      'ZEND' to identify next input as the final
Z depth.
<#      ZEND        Desired final depth.
<#  OUTPUT
<#      STEP        The (re)calculated actual step size
<#      NUMCUTS     The number of passes generated
<#
<#      NOTE: #ZWORK will have been updated to ZEND.
<#
<#  Declare input variables
      DECLARE TEXT &ONTO,&BNDRY,&CLOSED,&ROLL,&ZSTEP,&ZEND
      DECLARE DOUBLE STEP,ZEND,NUMCUTS
<#
<#  Declare internal variables
      DECLARE LOCATION @CURLOC
      DECLARE REAL ZWORK
      DECLARE DOUBLE I,
      DECLARE TEXT,&OPEN, &PROFILE
<#
<#  ***** START OF EXECUTABLE CODE *****
<#
<#
```

```
<#  Calculate the actual number of cuts and step size
<#
        NUMCUTS = AINT ( (ZWORK-ZEND) / STEP ) + 1
        If (((NUMCUTS-1)*STEP).GE.(ZWORK-ZEND)) NUMCUTS =
NUMCUTS - 1
        STEP    = (ZWORK-ZEND) / NUMCUTS
<#
<#  Add one more cut to account for first cut which doesn't
require
<#  a z-decrement
        NUMCUTS = NUMCUTS + 1
        PRINT
        PRINT MAKING  {NUMCUTS}  PASSES WITH Z INCREMENT =
{STEP}
        PRINT
<#
<#  Define an NCGROUP using #CURLOC to identify
side-of-boundary for
<#  'TO' option
<#
        !NCGROUP XYPNT LOC X  {XCOMP  (@CURLOC)} Y {YCOMP
(@CURLOC)};
<#  Create the text string for the profile command itself
<#

        &PROFILE = 'PROFILE' + &ONTO + ' ' + &BNDRY + ' ; '
       IF ( &ONTO(1) .EQ.'T' ) &PROFILE = &PROFILE + 'XYSIDE
XYPNT ; '
        IF ( & CLOSED(1).EQ.'C' ) &PROFILE = &PROFILE +
&CLOSED
       IF ( &ROLL(1).EQ.'R' ) &PROFILE = &PROFILE + &ROLL +
' '
<#
<#  If current location is not at ZWORK, plunge:
<#  IF (ZCOMP(@CURLOC) .NE. ZWORK) !PLUNGE
<#
<#  Loop on all cuts.
<#
        I = 1
        WHILE I  .LE. NUMCUTS
      I=I+1
      PRINT PASS # {I-1}  AT Z =  {ZWORK}
      ! &PROFILE
      !RETRACT XABS  {XCOMP(@CURLOC)}  YABS
{YCOMP(@CURLOC)} <#
      >ZABS  {ZCOMP(@CURLOC)}
<#
<#  Decrement ZWORK and plunge to next depth unless on last
cut
<#
```

```
        WHEN (I -1 .LT. NUMCUTS)
      ZWORK = ZWORK - STEP
      !PLANE ZWORK  {ZWORK}
      !PLUNGE
        ENDWHEN
        ENDWHILE
<#
<#  Send calculated step value and number of cuts back to the
JCF
<#
        APLINF/WRITE,6,1
        APLINF/WRITE,9,1
<#
<#  Done
<#
        RETURN
```

Constructing %VMACRO Grammar File Statements

%VMACRO statements link CVNC and a user-written CVMAC macro. %VMACRO provides more flexibility than %MACRO. Unlike %MACRO, %VMACRO defines a macro with a variable number of arguments. The %ARG symbol specifies the sequential number of the argument corresponding to the order of arguments in the PROC statement of the macro.

Please note: Up to 15 arguments are allowed in a PROC statement.

Use %ARG to represent any of the following with %VMACRO statements:

- Any CVNC reserved (major or minor) word.
- Numeric data. Numeric data is passed to the macro as double precision real values.
- Text data entered. Text data is passed to the macro as a CVMAC text string.
- Coordinate data entered. Coordinate data is passed to the macro as a CVMAC vector array.
- Entity selection data to be entered. Entity selection data is passed to the macro as a CVMAC entity array.

When using %ARG, the following rules apply:

- Argument sequence numbers may appear in any order.
- You can use the same argument more than once in a syntax statement, but you can assign only one value to that argument.
- If %ARG specifies a number greater than the number of arguments specified in the PROC statement, CVMAC treats it as an error condition and does not execute the macro.
- Not all arguments will be given values by the user input. The CVMAC construct APLINF/READ indicates such a condition by returning a zero for any argument not input. On this condition, your macro can assign an appropriate default.

Please note: You cannot use metasymbols or the %REPEAT and %MAV symbols in a %VMACRO statement.

Example

In this example, the REAM macro performs a reaming operation at a given series of coordinate locations. The PROC statement looks like this:

```
PROC REAM (%diam,&fit,@locs,@elocs,&notool,%tlnum)
```

Where

- %diam is the tool diameter
- &fit is the fitting type
- @locs are the locations of holes
- @elocs are the locations to specify the depth of the holes. You can specify an end location for every hole or you can input only one end location. If one location is input for @elocs, then depth is calculated from the Z-difference between @locs(i) and @elocs(1). If no locations are input for @elocs, then a default depth is used. (@locs and @elocs can also be input with a location NCGROUP.)
- &notool is a flag specifying no tool change
- %tlnum is the tool number for the CHGTOOL command

Please note: The CVMAC variables prefixed with the percent sign (%), ampersand (&), and at sign (@) have an assumed data type. See the *CVMAC Language Reference* for detailed information on variable names.

The definition of syntax for the REAM macro is

```
%VMACRO REAM DIAM %NEXP %ARG 1               <#
> %MGIVE (FITTYPE %TEXP %ARG 2)              <#
> %LIMIT (%XYZLOC) %FROM 1 %UPTO 300 %ARG 3  <#
> %MGIVE (ENDLOC %LIMIT (%XYZLOC)            <#
> %FROM 1 %UPTO 300 %ARG 4                   <#
> %CHOOSE (NOTOOL %ARG 5 %OR TOOLNUM %NEXP %ARG 6)
```

During CVNC execution, you can invoke this REAM macro by entering

```
NC:>REAM DIAM 0.25 ENDLOC: Model loc d d d; NOTOOL
```

The complete REAM macro is shown in the figure below.

REAM Macro

```
PROC REAM(%diam,&fit,@locs,@elocs,&notool,%tlnum)
<#
<#      GRAMMAR:
```

```
<#      %VMACRO REAM DIAM %NEXP %ARG 1 <#
<#      > %MGIVE (FITTYPE %TEXP %ARG 2) <#
<#      > %LIMIT (%XYZLOC) %FROM 1 %UPTO 300 %ARG 3 <#
<#      > %MGIVE (ENDLOC %LIMIT (%XYZLOC) <#
<#      > %FROM 1 %UPTO 300 %ARG 4 <#
<#      > %CHOOSE (NOTOOL %ARG 5 %OR TOOLNUM %NEXP %ARG 6)
<#
<#      Description:
<#      %diam           Tool diameter
<#      &fit            Fitting type
<#      @locs           Locations of holes
<#     @elocs         Locations to specify the depth of the
holes.
<#                      You can specify an end location for
every
<#                      hole or you can input only one end
location.
<#                      If one Location is input for @elocs
then
<#                      the depth is calculated from the
Z-difference
<#                      between @locs(i) and @elocs(1).
<#                      If nothing is input for @elocs then
a default<#                          depth is used.
<#      (@locs and @elocs can also be input with a location
ncgroup.)
<#
<#      &notool         Flag for not doing any tool change
<#      %tlnum          The tool number for the chgtool
command
<#
<#
        DECLARE LOCATION @locs(300),@elocs(300)
        APLINF/READ,3,nlocs
        APLINF/READ,4,nelocs
WHEN (&notool.EQ."")
<#
<#      Build the tool name
        &diam = %diam
        &toolnm = "REAM" + &diam + &fit
        &tlnum = %tlnum
<#
<#      If NOTOOL is not selected then calculate diameter
<#      of tool, then define and select it.
<#      (If FITTYPE is not specified then diam is not
changed)
<#
        IF (&fit.EQ."A") %diam = %diam + .0001
        IF (&fit.EQ."B") %diam = %diam + .0002
        IF (&fit.EQ."C") %diam = %diam + .0005
        IF (&fit.EQ.”D”) %diam = %diam + .0010
<#
<#      Define and select tool
<#        !! DELTOOL ”{&toolnm}”
          !! DEFTOOL ”{&toolnm}” MILL DIA {%diam}
          !! CHGTOOL {&tlnum} ”{&toolnm}”
ENDWHEN
<#
<#        Set default values for depth and clear etc.
<#        clear   is distance above hole for rapid tool
positioning
<#        cldist  is distance above hole to start cut feed
rate
<#        depth   is hole depth (usually calculated by
@elocs)
          clear = 1
          cldist = .2
          depth = 1
<#
          dis = clear - cldist
          &dis = dis
<#
REPEAT cmdloop:i=1,1,i.GT.nlocs
        <#
        <# Generate text string from location
        <# variable for move comand
        <#
        zloc = ZCOMP(@locs(i)) + clear
        @iloc = VECT(XCOMP(@locs(i)),YCOMP(@locs(i)),zloc)
        &iloc = ”X”
        &iloc = &iloc +
&VSTR(”F8.3,1X,’Y’,F8.3,2X,’Z’,F8.3”,@iloc)
        !! MOVE XYZLOC {&iloc} ;
        !! PLUNGE ZINC -{&dis}
        WHEN (nelocs.GT.1)
        <#
        <# Calculate depth for each hole from @elocs
        <#
           depth = ZCOMP(@locs(i)) - ZCOMP(@elocs(i))
        OR (nelocs.EQ.1)
           depth = ZCOMP(@locs(i)) - ZCOMP(@elocs(1))
        ENDWHEN
        &depth = depth
        !! CUT ZINC -{&depth}
        !! CUT ZINC {&depth}
        !! RETRACT XYZLOC {&iloc} ;
#cmdloop CONTINUE
RETURN
```

This example shows the ROUGH macro, which generates multiple rough cuts to a predefined NCGROUP. The PROC statement looks like this:

```
PROC ROUGH (&ONTO,&BNDRY,&CLOSED,&ROLL,
&ZSTEP,STEP,&ZEND,END,NUMCUTS)
```

The following %VMACRO grammar file statement describes the syntax for the ROUGH macro shown in the earlier figure, ROUGH Macro Using %MACRO:

```
<#  %VMACRO ROUGH                                <#
>  %MCHOOSE (ON %ARG 1 %OR TO %ARG 1)            <#
>  %TEXP %ARG 2                                  <#
>  %MCHOOSE (OPEN %ARG 3 %OR CLOSED              <#
>  %ARG 3) %MCHOOSE (ROLL %ARG 4 %OR NOROLL      <#
>  %ARG 4) STEP %VAR %ARG 5  ZEND %NEXP %ARG 6   <#
>  NUMCUTS %VAR %ARG 7
```

The syntax for ROUGH in CVNC might look like this:

```
NC:>ROUGH TO ‘SQUARE’ CLOSED ROLL ZSTEP STEP ZEND -2 NUMCUTS
```

This macro takes advantage of the %VMACRO and %ARG constructs. With %VMACRO it is possible to identify which phrases in the command line input are to be passed to which arguments in the macro’s PROC statement.

%VMACRO allows the use of %MCHOOSE and %MGIVE. The macro must use APLINF/READ to determine whether a particular argument has been provided in the command input line. If it has not, APLINF/READ will return a 0.

This syntax designed with %VMACRO is more flexible because

- It is not necessary to input ON/TO, OPEN/CLOSED or ROLL/NOROLL. The macro is structured to provide defaults using APLINF statements. In this case, the macro defaults to TO, CLOSED, and NOROLL.
- Although it is still necessary to type the modifier ZEND, it is not necessary for the macro to provide a corresponding argument, since this syntax does not assign ZEND to an argument number. It assigns the associated %NEXP to argument #6.

ROUGH Macro Using %VMACRO

```
PROC ROUGH (&ONTO,&BNDRY,&CLOSED,&ROLL,STEP,END,NUMCUTS)
<#  PURPOSE:
<#    This macro generates multiple rough cuts to a
predefined
<#    NCGROUP.  The first cut begins at the location of the
```

```
tool
<#   (called CURLOC) when the macro is called, and is made
at
<#   the current cutting depth (ZWORK). Subsequent cuts are
made
<#   at equal z-decrements (not to exceed STEP) until ZEND
is
<#   reached
<#
<#   Between passes, a retract to the XY of CURLOC, but at
the
<#   retract Z, is inserted, followed by a plunge to the
next
<#   depth.
<#
<#   After the last pass, only the retract move is issued.
ZWORK
<#   will have been reset to ZEND by this time, and the
macro
<#   also outputs the calculated values for the actual step
size
<#   and the number of cuts.
<#
<#  COMMAND SYNTAX                   CORRESPONDING ARGUMENTS
<#
<#  %VMACRO ROUGH
<#  > %MCHOOSE (ON %ARG 1 %OR TO %ARG 1)     <#
<#  > %TEXP %ARG 2      <#
<#  > %MCHOOSE (OPEN %ARG 3 %OR CLOSED      <#
<#  > %ARG 3) %MCHOOSE (ROLL %ARG 4 %OR NOROLL    <#
<#  > %ARG 4) STEP %VAR %ARG 5  ZEND %NEXP %ARG 6 <#
<#  > NUMCUTS %VAR %ARG 7
<#
<#  Description of arguments
<#  INPUT
<#     &ONTO  =  'ON' or 'TO' to describe desired profile
mode.
<#           Default = 'TO'.
<#           NOTE: CURLOC should be properly positioned with
<#           respect to the first boundary element to
             correctly identify side-of-boundary for ON.

<#    &BNDRY  =  Name of NCGROUP to be profiled.  It must be
comprised
<#           of at least 3 entities.  It is passed into
<#           this macro as a text string (rather than
<#           variable or entitygroup) solely to facilitate
<#           the creation of thecorrect text for the profile
             commands to be issued.
<#
```

```
<#     &CLOSE  'OPEN' or 'CLOSED' to describe the boundary.
<#         Default = CLOSED.
<#
<#     &ROLL'ROLL' or 'NOROLL' to describe radial motion
<#           around corners or not. Default = NOROLL.
<#
<#     STEPThe maximum step size - input a variable, so
<#           that the actual (re)calculated step size may be
             output.
<#     ZENDDesired final depth.
<#  OUTPUT
<#     STEP  The (re) calculated actual step size
<#     NUMCUTS  The number of passes generated.
<#     NOTE: #ZWORK will have been updated to ZEND.

<#  Declare input variables
        DECLARE TEXT &ONTO,&BNDRY,&CLOSED,&ROLL,&ZSTEP,&ZEND
        DECLARE DOUBLE STEP,ZEND,NUMCUTS
<#
<#  Declare internal variables
        DECLARE LOCATION @CURLOC
        DECLARE REAL ZWORK
        DECLARE DOUBLE I, NUMBER
        DECLARE TEXT, &PROFILE
<#
<#  ***** START OF EXECUTABLE CODE *****
<#
<#  Get current location and current ZWORK
<#
      APLSYSR/'#CURLOC',@CURLOC
      APLSYSR/'#ZWORK',ZWORK

<#  Check for inconsistent input (ZWORK less than ZEND)
<#
      WHEN (ZWORK.LT.ZEND)
     PRINT ZWORK  =  {ZWORK} IS LESS THAN ZEND  =  {ZEND}
     ABORT
      ENDWHEN
<#
<#  If no input for &ONTO, &CLOSED, or &ROLL, set them to
<#  default conditions.
<#
      APLINF/READ, 1, NUMBER
      If (NUMBER.EQ.0) &ONTO ='TO'
      APLINF/READ, 3, NUMBER
      If (NUMBER.EQ.0) &CLOSED = 'CLOSED'
<#
      APLINF/READ, 4, NUMBER
      If (NUMBER.EQ.0) &ROLL = 'NOROLL'
<#
```

```
<#  Calculate the actual number of cuts and step size
<#
      NUMCUTS = AINT ( (ZWORK-ZEND) / STEP ) + 1
      If (((NUMCUTS-1)*STEP).GE.(ZWORK-ZEND)) NUMCUTS =
NUMCUTS - 1
      STEP    = (ZWORK-ZEND) / NUMCUTS
<#
<#  Add one more cut to account for first cut which doesn't
require
<#  a z-decrement
      NUMCUTS = NUMCUTS + 1
      PRINT
      PRINT MAKING  {NUMCUTS}  PASSES WITH Z INCREMENT =
{STEP}
      PRINT
<#
<#  Define an NCGROUP using #CURLOC to identify
side-of-boundary for
<#  'TO' option.

<#      !NCGROUP XYPNT LOC X  {XCOMP  (@CURLOC)} Y {YCOMP
(@CURLOC)};
<#

<#  Create the text string for the profile command itself
<#
      &PROFILE = 'PROFILE' + &ONTO + ' ' + &BNDRY + ' ; '
      IF ( &ONTO(1) .EQ.'T' ) &PROFILE = &PROFILE + 'XYSIDE
XYPNT ; '
      IF ( &CLOSED(1).EQ.'T') &PROFILE = &PROFILE + &CLOSED

    IF ( &ROLL (1).EQ.'R') &PROFILE = &PROFILE + &ROLL + ' '

<#
<#  If current location is not at ZWORK, plunge:
<#
      IF (ZCOMP(@CURLOC) .NE. ZWORK) !PLUNGE
<#
<#  Loop on all cuts.
<#
      I = 1
      WHILE I  .LE. NUMCUTS
     I=I+1
     PRINT PASS # {I-1}  AT Z =  {ZWORK}
     ! &PROFILE
     !RETRACT XABS  {XCOMP(@CURLOC)}  YABS  {YCOMP(@CURLOC)}
<#  Decrement ZWORK and plunge to next depth unless on last
cut
<#
      WHEN (I -1 .LT. NUMCUTS)
      ZWORK = ZWORK - STEP
      !PLANE ZWORK  {ZWORK}
      !PLUNG
       ENDWHEN
       ENDWHILE
<#
<#  Send calculated step value and number of cuts back to
the JCF
<#
    APLINF/WRITE,5,1
    APLINF/WRITE,7,1
<#
<#  Done
<#
      RETURN
```

Specifying Your Macro Library

Use the MACLIB command to specify the macro library containing the macro or macros you want to invoke.

Using MACLIB

To use MACLIB, enter

```
NC:>MACLIB 'libraryname'
```

Where

'libraryname' is the relative path name of the library containing the macro(s) you are going to use. This library is the same that you designated with the MACROLIB statement in the CVMAC link file.

You may specify up to four macro libraries. CVNC searches each library sequentially to find a macro. If more than one library is specified, a macro is written in the first library name specified.

Please note: The default macro library is `cam.{ncturn,ncmill,nc3ax,nc5ax,fab}.macro.lib`. You can configure CVNC so that it defaults to a macro library containing your macro(s), eliminating the need to specify the macro library in the JCF. This is described later in this chapter.

Example

To specify the macro library for the INSURF macro, enter

```
NC:>MACLIB 'MACRO.MACLIB'
```

Check Macros

A check macro library is a set of procedures (one is allowed per CVNC command) which automatically executes when a CVNC command bearing the same name as a macro in the check macro library is issued.

Check macros check to see that parameters and conditions are not violated during a command's execution. For example, you can define a check macro for the FEED command to check that feed rate does not exceed 100 IPM. When FEED is executed, its check macro is automatically invoked. The check macro can be written to display an error message and terminate processing of the FEED command if FEED exceeds 100 IPM, as in the following example:

```
PROC feed
APLSYSR/"#FEDRAT", FEDRAT, "#FEDUNIT", &UNITS
WHEN (&UNITS.EQ "IPM" .AND. FEDRAT .GT. 100)
PRINT feed rate out of bounds (<100)
ABORT
ENDWHEN
```

Check macros, otherwise invisible in the JCF, display a warning message if any check conditions are violated.

Please note: When you create a check macro, enter the procedure name (such as "feed") in lowercase. For example, the first line of your macro should appear as "PROC feed". When you use the CHECKLIB command to invoke a new check macro, CVNC checks for uppercase characters in the PROC statement of the macro. If it finds uppercase letters, it displays an error message:

```
CHECK LIBRARY CONTAINS A MACRO WITH UPPER CASE LETTERS IN
NAME
```

After you have written, compiled, and linked a check macro, access it within CVNC with the CHECKLIB command.

Creating a Check Macro

Write the text file for a check macro, and then compile and link it. To create a check macro,

1. Create the macro text file using any editor.
2. Compile the macro with cvmcomp.
3. Create a link file for the macro.
4. Link the macro with cvmlink.

There is a utility, checkutil, which takes you through steps 1-4 with a series of prompts. This is described later in this section.

Example: This example creates a check macro for the FEED command. If feed exceeds the boundary set in the check (100 IPM), an error message will be displayed and the command will be terminated.

1. Create the macro text file. The following text file is named `mycheck/feed.`

```
PROC feed
APLSYSR/“#FEDRAT”, FEDRAT, “#FEDUNIT”, &UNITS
WHEN (&UNITS.EQ ”IPM” .AND. FEDRAT .GT. 100)
PRINT feed rate out of bounds (<100)
ABORT
ENDWHEN
```

2. Compile the macro.

   ```
   % cvmcomp mycheck.feed
   ```

3. Create a link file for the macro. The following link file is named `mycheck/link.`

   ```
   MACROLIB mycheck.lib
   LINKM mycheck.feed
   ENDLINK
   ```

4. Link the macro.

   ```
   % cvmlink mycheck.link
   ```

Please note: Check macros do not require you to append a statement to the grammar.

Specifying Your Check Macro Library

Use the CHECKLIB command to specify the macro library containing the check macro or macros you want to invoke. CHECKLIB specifies a name of a macro library containing the macro or macros you want to use, and turns the execution of check macros ON or OFF. When CHECKLIB is ON, CVNC processes a command and then executes the CVMAC macro with the same name (if there is one). If CHECKLIB is OFF, CVNC ignores any check macros.

Using CHECKLIB

To use CHECKLIB, enter

```
NC:>CHECKLIB ‘libraryname’
```

Where

‘libraryname’ is the relative path name of the library containing the macro(s) you are going to use. This directory is the same that you designated with the MACROLIB command in the CVMAC link file. Only one check macro library is allowed in a JCF.

Please note: The default macro library is `cam.{ncturn,ncmill,nc3ax,nc5ax,fab}.check.lib`.

You can configure CVNC so that it defaults to a macro library containing your macro(s), eliminating the need to specify the macro library in the JCF. This is described later in this chapter.

Example: To specify the check macro library for the mycheck/feed macro, enter

```
NC:> CHECKLIB ‘MYCHECK.LIB’
```

Now whenever FEED is executed, the check macro executes. If the cut feed rate is out of bounds (>100), the error message will print and the invalid FEED command will be terminated. CVNC will reposition the attention marks at the offending line just as if CVNC had encountered an error in processing.

Using the checkutil Command

checkutil is an operating system-level command for creating check macros. This utility

- Creates CVMAC code for CVNC check macros
- Compiles and links macros into one library
- Allows you to add or modify macros in an existing library
- Allows you to choose the directory where the check library will live
- Allows you to name the check library
- Provides online help for checkutil and for constructing a check macro

checkutil uses the system variables of a command to check that conditions are being met. After you specify to checkutil the command for which you want to create a check, it lists the associated system variables.

For a comprehensive list of system variables, see Appendix D of the *CVNC System User Guide and Menu Reference*, or access the online documentation for system variables by entering the following:

```
NC:>#!
```

Example

The following example creates a check macro for the SPEED command.

To use checkutil, follow this procedure:

1. Enter

```
%checkutil
```

2. Press RETURN to begin, or type ‘h’ for ‘help’ documentation that explains the checkutil procedure.

```
CHECKUTIL
Type ‘h‘ to see documentation or <CR> to begin
=>
```

3. Choose the directory where the check library will live and specify a name for the library. The default directory is your current working directory; the default library is chklib.

```
What directory do you want the check library to be in? (<CR>
for current. Start with / if you are giving full path name.)
=>
```

```
Using directory /usr2/cadds/mcknight/check
Enter the check library name you want to use, <CR> for
default (chklib)
=>
```

4. Specify the command for which you want to make a check macro. The system variables associated with this command will be listed.

```
What command do you wish to develop a check rule for? (<CR>
to begin linking)

=>speed

The system variables associated with this command are:
(The @ and & indicate coordinate or text variables, others
are numeric)
#1          #&RANGE
#2          #&SPINDIR
#3          #SPINSPD
#4          #&SPINSTA
#5          #&SPINUN
```

5. Type in the CVMAC check condition and the check condition message. ‘Help’ documentation is available on construction of CVMAC conditional statements.

```
Type CVMAC check condition no (1) for SPEED; type ‘h‘ to see
example constructs, ‘t‘ to see the system variable list
again, <CR> to continue.

=>when #3.gt.300

WHEN SPINSPD.GT.300
Type message for violation of check condition no (1):
=>Hey, the spindle speed is too fast!

The system variables associated with this command are:
(The @ and & indicate coordinate or text variables, others
are numeric)
#1          #&RANGE
#2          #&SPINDIR
#3          #SPINSPD
#4          #&SPINSTA
#5          #&SPINUN
Type CVMAC check condition no (2) for SPEED; type ‘h‘ to see
example constructs, ‘t‘ to see mav list again, <CR> to
continue.
```

6. Press RETURN to create the CVMAC source file, and display it on the screen.

```
=>
Creating CVMAC source file ...
```

```
This is speed:
PROC speed
APLSYSR/'#SPINSPD',spindspd
WHEN SPINSPD.GT.300
Print Hey, the spindle speed is too fast!
ABORT
ENDWHEN
RETURN
```

7. Press RETURN to compile the file. If necessary, you can edit the file with the **vi** editor before compiling it.

```
Type 'e' to edit file (vi editor) or <CR> to compile
=>
Compiling CVMAC source file ...
```

8. Press RETURN to link the file or create another check macro.

```
What command do you wish to develop a check rule for? (<CR>
to begin linking)
=>
Linking check library ...
```

The check macro is now in the check library `/usr2/cadds/mcknight/check/chklib.` If CHECKLIB is on, this macro will execute whenever SPEED is executed.

Output Macros

You can write CVMAC macros to customize CVNC output generation. Output macros customize output for CVNC commands that create machine control statements (pass-through statements).

If the output macro contains pass-through statements that are not in the grammar, append them to a copy of the grammar with ncgram append.

For more information on pass-through statements, see Chapter 3, “Customizing with Pass-through Statements.” For more information on ncgram append, see Chapter 2, “Using CVNC Grammar Files.”

Please note: CVNC-P2 provides several output macro commands, but you can still create your own output macros for CVNC-P2. For more information on CVNC-P2 output macro commands, see the *CVNC-P2 User Guide* and the *CVNC-P2 Command Reference*.

After writing, compiling, linking, and appending the macro to the grammar, access it within CVNC using the OUTLIB command. OUTLIB allows you to turn the generation of output macros ON or OFF.

Creating an Output Macro

To create an output macro, write, compile, and link the CVMAC text file as follows:

1. Create the macro text file using any editor.
2. Compile the macro with cvmcomp.
3. Create a link file for the macro.
4. Link the macro with cvmlink.

Example: This example creates an output macro that customizes the output for the SPINDL command.

1. Create the macro text file. The following displays the text file `myoutput/spindl.`

```
PROC speed
<# output macro to reformat SPINDL output for CVNC.
APLSYSR/'#RANGE', &range
rval = 2
IF (&range .EQ. 'HIGH') rval = 1
IF (&range .EQ. 'MEDIUM') rval = 2
IF (&range .EQ. 'LOW') rval = 3
IF (&range .EQ. 'AUTO') rval = 4
.
.
.
END<#
APLSYSR/'#SPINSTA', &onoff
<#
WHEN (&onoff .EQ. 'OFF')
! SPINDL OFF
<#
ELSE
APLSYSR/'#SPINSPD',spindle
APLSYSR/'#SPINDIR',&dir,'#SPINUN', &unit
<#
!SPINDL {spindle} {&unit} {&dir} RANGE {rval}
<#
ENDWHEN
<#
RETURN
<#
END
```

2. Compile the macro.

```
% cvmcomp myoutput.spindl
```

3. Create a link file for the macro. The following link file is named `myoutput/link.`

```
MACROLIB myoutput.lib
LINKM myoutput.spindl
ENDLINK
```

4. Link the macro.

```
% cvmlink myoutput.link
```

5. In this example, it is necessary to append the following statement to the grammar, using ncgram append:

```
%STMT SPINDL %CHOOSE ( OFF %OR %NEXP RPM %CHOOSE
<#   (CLW %OR CCLW) RANGE %NEXP )
```

Please note: To append a pass-through statement to the grammar, you can use the macutil command (described earlier), which takes you through steps 1-5.

Specifying Your Output Macro Library

OUTLIB specifies the directory of output macros to be accessed for output generation of CVNC commands that create machine control statements. OUTLIB allows you to turn execution of output macros ON or OFF.

When OUTLIB is ON, CVNC activates the last specified macro library for output and switches into customized output mode.

When OUTLIB is OFF, default output is generated for all CVNC commands.

Using OUTLIB

To use OUTLIB, enter

```
NC:>OUTLIB ‘libraryname’
```

Where

‘libraryname’ is the relative path name of the library containing the macro(s) you are going to use. This file is the same that you designated with the MACROLIB command in the CVMAC link file. Only one output macro library is allowed in a JCF.

Please note: The default macro library is `cam.{ncturn,ncmill,nc3ax,nc5ax,fab}.output.lib`.

You can configure CVNC so that it defaults to a macro library containing your macro(s), eliminating the need to specify the macro library in the JCF. This is described later in this chapter.

Examples

The default CVNC APT syntax for SPINDL is

```
SPINDL/RPM, 23000, CLW, RANGE, AUTO
```

When the library containing the myoutput/spindl macro is activated with the command

```
OUTLIB ‘MYOUTPUT.LIB’
```

The following output is generated:

```
SPINDL/23000,RPM, CLW, RANGE, 4
```

Range defaults to 2 (medium) even if you did not enter a RANGE, as follows:

```
Input:SPEED RPM 3000 CCLW
Output:SPINDL/3000,RPM,CCLW,RANGE,2
```

CVNC allows numeric ranges in the SPEED command. However, the above macro allows you to input LOW, MEDIUM, etc.

Point Macros

Point macros, by passing text to output, allow you to control the machine tool during tool path generation. The following commands contain several predefined points.

- AREAMILL [M2]
- SURFCUT3 [M3]
- SURFCUT4 [M5 Four Axis]
- SURFCUT5 [M5]

Predefined points are defined points during command execution. Each predefined point defined for the AREAMILL and SURFCUT commands has a unique name (see the individual application documentation for a complete list of predefined points).

For each of these points, a spot is reserved for a macro by the same name. For example, the point before a retract to the CLEAR plane in an initial AREAMILL plunge is called AMIL05. A macro can be written for this point; it must be named amil05. Note that the macro name must be in lowercase.

Whenever a predefined point is reached during command execution, CVNC looks for a macro sharing the name of that point. If there is one, it is executed; if not, processing continues.

Creating a Point Macro

To create a point macro, write, compile, and link the CVMAC text file as follows:

1. Create the macro text file using any editor.
2. Compile the macro with cvmcomp.
3. Create a link file for the macro.
4. Link the macro with cvmlink.

Contents of Point Macros

Point macros can contain CVMAC code to perform a variety of activities such as

- Eliciting input from the operator
- Reading or writing from files
- Reading or writing from the CADDS data base
- Performing algebraic calculations

The following are not allowed in point macros:

- CVNC commands (except TEXTOUT or pass-through statements)
- CVMAC geometry commands
- Macro calls and command files
- Arguments

Point macros do not automatically generate output. To generate output from a point macro, you can

- Use the TEXTOUT [SYS] command in a macro to insert a text string into the output. This is described in the next section.
- Include a pass-through statement in the macro. If this statement is not in the grammar, append it to a copy of the grammar. For more information on pass-through statements, see Chapter 3, “Customizing with Pass-through Statements.”

Generating Output with Point Macros

To generate output with point macros, the macro must contain either TEXTOUT statements or CVNC pass-through statements.

Using TEXTOUT in a Point Macro

You can insert a TEXTOUT command in a point macro to pass a text string directly to an APT or COMPACT output file. You can also use TEXTOUT to generate a binary CLF file record.

Example

The following macro inserts the text string ‘SPINDL 2000 RPM’ into the output at the point AMIL05:

```
PROC amil05
.
.
!TEXTOUT “SPINDL 2000 RPM”
END
```

Using Variables with TEXTOUT

You can assign a variable to a predefined point. These variables hold the text you want to output at that point. To set the variable, use the ASSIGN OUTPUT command [SYS]. Variables allow you to change the text output without modifying the macro.

Examples

In this example, TEXTOUT outputs the contents of variable *a* at the point AMIL05. Variable *a* is defined in CVNC with the following command:

```
ASSIGN OUTPUT %AMIL05 ‘SPINDL 2000 RPM’
PROC amil05
DECLARE TEXT a
APLSYSR/“%AMIL05”,a
!TEXTOUT “{a}”
END
```

In the next example, the following commands and the macro (see display) insert the statements ‘SPINDL 2000 RPM’ and ‘COOLNT MIST’ before every DIRECT positioning move in the output:

```
ASSIGN OUTPUT %AMIL05 ‘SPINDL 2000 RPM ! COOLNT MIST’
AREAMILL BOUND $L1 $L2 $L3 $L4;
```

Point Macro

```
<#  This macro converts a ! delimited string in %AMIL05,
passed from CVNC,into separate
<# pass-through output statements
<#
<#
     PROC amil05
     DECLARE TEXT &OUTSTR (255)
     DECLARE REAL A, B
<#
<#  Get a string from CVNC and add a final ! delimiter;
<#  initialize substring pointers A and B to the
<#  first substring
     APLSYSR/”%AMIL05”,&OUTSTR
     A = 1
     &OUTSTR = &OUTSTR + “!”
     B = FNDB(“!”,&OUTSTR)
     WHILE B > A<#
<#  Output string
<#
     !TEXTOUT  “{&OUTSTR(A,B)}”
<#  Set A and B to the next substring
<#  Note that B > A only if a valid substring is found
<#
     A = FNDA (“!”,&OUTSTR(B,))+B-1
     B = FNDB (“!”,&OUTSTR(A,))+A-1
    ENDWHILE
    RETURN
    END
```

Specifying Your Point Macro Library

Use the MACLIB command to specify the macro library containing the point macro or macros you want to invoke.

Using the MACLIB Command

To use MACLIB, enter

```
NC:>MACLIB ‘libraryname’
```

Where

‘libraryname’ is the relative path name of the library containing the point macro(s) you are going to use. This library is the same that you designated with the MACROLIB statement in the CVMAC link file.

You may specify up to four macro libraries. CVNC searches each library sequentially to find a macro. If more than one library is specified, the macro is written in the first library name specified.

Please note: The default macro library is `cam.{ncturn,ncmill,nc3ax,nc5ax,fab}.macro.lib`.

You can configure CVNC so that it defaults to a macro library containing your macro(s), eliminating the need to specify the macro library in the JCF. This is described later in this chapter.

You can specify a command macro library and a point macro library in the same MACLIB command. For example,

```
NC:>MACLIB ‘pointmaclib’ ‘commandmaclib’
```

Example

To specify the macro library for the point macro amil05, enter

```
NC:>MACLIB ‘MACRO.RESERVELIB’
```

Configuring CVNC Macro Libraries

You can configure CVNC to default to the macro library containing your macro(s). This eliminates the need to specify the macro library within a JCF.

Default Libraries

For each type of macro, there is a default macro library, as listed in the following table:

Table 4-5 Default Macro Libraries

Type	Default Library
Command	cam.{ncturn,ncmill,nc3ax, nc5ax,fab}.macro.lib
Check	cam.{ncturn,ncmill,nc3ax, nc5ax,fab}.check.lib
Output	cam.{ncturn,ncmill,nc3ax, nc5ax,fab}.output.lib
Point	cam.{ncturn,ncmill,nc3ax, nc5ax,fab}.macro.lib

There are two ways you can configure CVNC so that it defaults to a library containing your macros:

- Place a library, with the same name as the default library, earlier in the CVPATH.
- Overwrite the original library.

Placing a Library Earlier in the CVPATH

You can create a library in a directory earlier in your CVPATH than the default libraries. CVNC will use the library that it finds first.

The default macro libraries (listed on the previous page) are located in `/usr/apl/cadds/data/cam.` If you create a library with the same path as the default library earlier in your CVPATH, CVNC will find that library first and use it, rather than the default.

Example: In the following CVPATH, the `/usr2/cadds` directory is located before `/usr/apl/cadds/data/cam/ncmill` (that contains macro.lib, check.lib, etc.). A library named macro.lib is created in the directory `/usr2/cadds/cam/ncmill.`

```
CVPATH‘/USR2/CADDS:/USR/APL/CADDS/DATA/MODTAB:
/USR/APL/CADDS/DATA:  /USR2/CADDS/PARTS=C:.:/USR/APL/CADDS:
/USR/APL/CADDS/DATA/DMENU:/USR/APL/CADDS/DATA/VNTAB:
/USR/APL/CADDS/DATA/CAM/NCMILL’
```

CVNC defaults to the maclib.lib library it finds first, therefore it defaults to `/usr2/cadds/cam/ncmill/macro.lib.`

Overwriting the Original Library

You can overwrite the default libraries located in

```
/usr/apl/cadds/data/cam/{ncmill,ncturn,nc3ax,nc5ax,fab)
```

To do this,

1. Make a copy of the default library you want to reconfigure. This copy serves as a backup so that you can convert back.
2. Overwrite the existing library by copying a library you have created to it.

Example

In the following example, a library named `mymacros`, located in the macro directory, is copied into the default macro library for command and point macros.

1. Copy the original library to the `librarysave` directory.

   ```
   %cp /usr/apl/cadds/data/cam/ncmill/macro/lib librarysave
   ```

2. Overwrite the existing library by copying `mymacros` to it.

   ```
   %cp macro/mymacros
   /usr/apl/cadds/data/cam/ncmill/macro/lib
   ```

Now, any macro linked to the `mymacro` library can be invoked, without issuing the MACLIB command.

Modifying CVNC-supplied Command Macros

CVNC provides several command macros. You can modify any of these macros by editing the macro source file and the append source file.

To modify CVNC-supplied macros,

1. Edit the macro text file located under

   ```
   /usr/apl/cadds/data/cam/nc{mill,turn,fab}/grammar/macro
   ```

Depending on the modification made to the macro, you may have to

2. Edit the append-mac file for that macro. The append-mac file is the source file containing the %MACRO and %VMACRO grammar file statements for CVNC-supplied macros.

When to Edit the append-mac File

If there is any modification to a macro's text file that makes the grammar file statement incorrect, edit the append-mac file. For example, the following PROC statement is in the text file for the macro DPOCK:

```
PROC DPOCK (&APPR,&LACE,&ONTAN,&CLDST,CLDST,
&XYZLOC,@DIG1)
```

The grammar file statement in the append-mac file is

```
%MACRO DPOCK %DOC 'DPOCK' %CHOOSE (APPRX %OR APPRY)
<#            %CHOOSE (LACEX %OR LACEY)
<#   %CHOOSE (ON %OR TANTO) CLDST %NEXP XYZLOC
<#   %LIMIT (%XYZLOC)        %FROM 2 %UPTO 2
```

If the &ONTAN argument is removed from the PROC statement (and corresponding CVMAC code), so that it reads

```
PROC DPOCK (&APPR,&LACE,&CLDST,CLDST,&XYZLOC,@DIG1)
```

the grammar file statement must be changed to the following:

```
%MACRO DPOCK %DOC 'DPOCK' %CHOOSE (APPRX %OR APPRY)
<#            %CHOOSE (LACEX %OR LACEY)
<#          CLDST %NEXP XYZLOC %LIMIT (%XYZLOC)
<#           %FROM 2 %UPTO 2
```

When Not to Edit the append-mac File

If modification to a macro's text file does not affect the grammar file statement, you do not need to edit the append-mac file. For example, you can add a PRINT statement to the code in the following text file:

```
<# select lace direction now
IF (&LACE (5) .EQ. 'X') GOTO 20
<#
<# y-lace calculations
<#
DELTN=-DELTYT
!CUT YINC {DELTYT}
IF (ISTEP .EQ. 1) GOTO 100
It now reads
<# select lace direction now
IF (&LACE (5) .EQ. 'X') GOTO 20
<#
<# y-lace calculations
<#
PRINT LACE DIRECTION = {&LACE}
DELTN=-DELTYT
!CUT YINC {DELTYT}
IF (ISTEP .EQ. 1) GOTO 100
```

This PRINT does not affect any data requested by the grammar file statement or the PROC statement. You do not need to edit the append-mac file.

Editing the Macro Text File

The table on the next page lists CVNC-supplied macros and the directories where macro text files are stored. For each macro, there is a text file and two binary files (macro.cm and macro.dm). To modify a macro, edit, compile, and link the text file. Appendix C, "Pass-through Statements and Macros," presents several of these text files.

Procedure

To edit a macro text file for a CVNC-supplied command,

1. Make a backup copy of the original macro for safety purposes. For example, if you want to modify the CVNC-M2 macro DPOCK, enter

```
%cp /usr/apl/cadds/data/cam/ncmill/macro/dpock savedpock
```

2. Edit the original text file (not your backup copy) with any editor. For example:

```
%cd /usr/apl/cadds/data/cam/ncmill/macro
%vi dpock
```

3. Compile the macro with cvmcomp.

```
%cvmcomp dpock
```

4. Link the macro to a macro library with cvmlink. You do not need to write a link file. Under the macro directory, there is a link file called link. Enter

```
%cvmlink link
```

You may need to edit the append-mac file and append it to the grammar using ncgram append. This is described in the next section.

Table 4-6 CVNC-supplied Macros

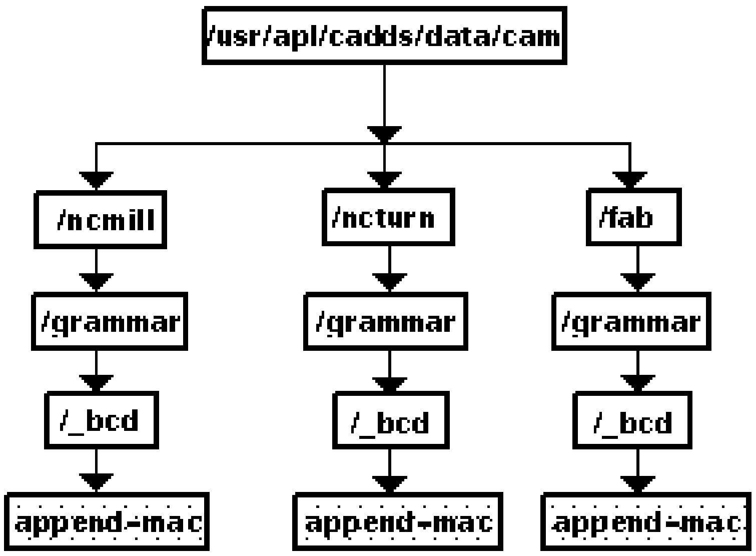
Macro	Product	Directory of Macro Text File Relative to /usr/apl/cadds/data/cam
INSARC3	CVNC-M2, -T2, -P2	{ncmill,ncturn,fab}/ grammar/macro
INSFIL	CVNC-M2, -T2, -P2	{ncmill,ncturn,fab}/ grammar/macro
INSCIR	CVNC-M2, -T2, -P2	{ncmill,ncturn,fab} /grammar/macro
INSPOI	CVNC-M2, -T2, -P2	{ncmill,ncturn,fab} /grammar/macro
DELENT	CVNC-M2, -T2, -P2	{ncmill,ncturn,fab}/ grammar/macro
LEGROUP	CVNC-M2, -T2, -P2	{ncmill,ncturn,fab}/ grammar/macro
VERENT	CVNC-M2, -T2, -P2	{ncmill,ncturn,fab}/ grammar/macro
TLCHG	CVNC-M2	ncmill/grammar/macro
SPROF	CVNC-M2	ncmill/grammar/macro
DPROF	CVNC-M2	ncmill/grammar/macro
DPOCK	CVNC-M2	ncmill/grammar/macro
FTURN	CVNC-T2	ncturn/grammar/macro
RFACE	CVNC-T2	ncturn/grammar/macro
GROOVE	CVNC-T2	ncturn/grammar/macro
BLEND	CVNC-T2	ncturn/grammar/macro
GRID	CVNC-P2	ncturn/grammar/macro

Editing the append-mac File

The append-mac file contains grammar file statements for CVNC-supplied macros; it is a text file and can be edited. If you modified a macro's text file so that the grammar file statement is incorrect, edit the append-mac file. There are four steps to editing the append-mac text file containing the CVNC-supplied macro you want to modify. These steps are on the next page.

There is an append-mac file for CVNC-M2, CVNC-T2, and CVNC-P2. CVNC-M5 and CVNC-M3 can use CVNC-M2 macros. The following chart shows where these append text files are located on the system. Appendix C, "Pass-through Statements and Macros," shows the contents of these append source files.

Figure 4-1 Location of Append Text Files



Procedure

To edit the append-mac file, follow these four steps.

1. Make a directory to store a copy of the append text file and a copy of the Base grammar.

```
% mkdir my my/grammar my/grammar/_bcd
```

2. Copy the append-mac file to your grammar/_bcd directory.

For example, to edit the DPOCK macro, enter the following:

```
%cp /usr/apl/cadds/data/cam/ncmill/grammar/_bcd
   /append-   mac my/grammar/_bcd
```

3. Make a copy of the Base binary files.

```
%cp /usr/apl/cadds/data/cam/ncmill/grammar/base
 {_voc,_grm}my/grammar
```

Please note: Use the base grammar files when appending the append-mac file. If you use the default grammar files or the COMPACT II grammar files, the append file has already been appended. CVNC will not allow you to append the same file twice.

In your my/grammar directory, you should now have the following directories and files:

```
_bcd/append-mac
_voc
_grm
```

4. Edit the append-mac file with any editor.

```
%vi my/grammar/_bcd/append-mac
```

Next, append the append-mac file to your copy of the grammar using ncgram append. Remember that the Base grammar does not include macros or pass-through statements. In this procedure, you appended all the macros. To use CVNC-supplied pass-through statements, you must also append the append-post or append-compact file. For more information on using ncgram append, see Chapter 2, “Using CVNC Grammar Files.”

Appendix A

Entity Types List

The table in this appendix lists all the CADDS entity types, along with the corresponding entity type numbers. These numbers are necessary when using the %VALID and %INVALIrammar file statement symbols.

- Entity Types List

Entity Types List

Table A-1 Entity Types

Number	Entity Name
2	Point
3	Line
5	Arc
6	Conic
7	Spline
8	B-spline
9	Curve Pole (Cpole)
10	Curve Point (Cpoint)
11	Surface Point (Spoint)
12	Nspline
14	Nsurface
16	Tabulated Cylinder (Tcylinder)
17	Surface of Revolution (Srevolution)
18	Ruled Surface (Rsurface)
19	B-spline Surface (Bsurface)
20	Surface Pole (Spole)
21	Scalar
22	Vector
24	Shape
25	Rectangle
31	Crosshatch
32	Feature Control Symbol (FCS) and Flagnote
33	Linear/Ordinate Dimension (Ldimension/Odimension)
34	Angular Dimension (Adimension)
35	Radial Dimension (Rdimension)
36	General Label
37	Diameter Dimension (Ddimension)
50	Profile Tool Path
51	Pocket Tool Path
52	Regenerative MABS Tool Path
53	Point-to-point Tool Path
54	Surface Machining Tool Path
55	Surface Intersection Tool Path
58	Draft Curve
70	String

Number	Entity Name
71	Nodal Line (Nline)
73	Finite Element Modeling (FEM) Element
74	Grid Point (Gpoint)
75	Mass Point (Mpoint)
76	Contour Map
77	Reserved for FEM
78	Reserved for FEM
80	Subfigure Instance
81	Connect Node (Cnode)
82	Text Node (Tnode)
83	Nodal Figure (Nfigure) Instance
85	Text
86	Nodal Text (Ntext)
87	Relation
88	Plane/Face/Hole
90	Subassembly Instance
91	Solid or Trimmed Surface
92	Face
93	Edge
94	Vertex
95	Reserved for Solid Modeling
96	Reserved for Solid Modeling
100-200	Reserved
201-220	Available for users
250	Part Parameter Entity

Appendix B

CADDS/UNIX File Name Character Conventions

Depending upon which commands you are using, you must specify either directory names with CADDS or UNIX file name character conventions. This appendix lists the CADDS characters and the equivalent UNIX characters and provides examples and samples to demonstrate the conventions.

CADDS/UNIX File Name Character Conventions

While using the following commands, you must specify directory names with CADDS file name character conventions:

- ncgram append
- cvmlink
- cvmcomp
- seqlist
- sequence_output
- nclinker

While using the following commands, you must specify directory names with UNIX file name character conventions:

- macutil
- checkutil

The table below lists the CADDS file name characters and the equivalent UNIX character.

Table B-1 File Name Character Conventions

CADDS Character	UNIX Character
=	**/** (leading / in path only)
.	/
&	_ (underscore)
@	.

Example: `% cvmcomp =usr.alison.macro.verent@one` looks for the UNIX file `/usr/alison/macro/verent.one` or `/usr/alison/macro/_bcd/verent.one.`

Example: `% cvmlink =usr.alison.macro.linkfile` looks for the UNIX file `/usr/alison/macro/linkfile` or `/usr/alison/macro/_bcd/linkfile.`

Using cvmcomp: Example

```
%cvmcomp macro.insurf

UNDECLARED SYMBOLS


$ent


CODE MODULE SIZE  =  52
DATA MODULE SIZE  =  40
BUFFERED DATASIZE =  40
VIRTUAL DATASIZE  =  0

macro.insurf@cm FILED
macro.insurf@dm FILED
```

Using cvmlink: Example

```
%cvmlink macro.linkfile

CMEM IMAGE FILE macro.maclib@cm!
00000200 WORDS WRITTEN TO IMAGE
DMEM IMAGE FILE macro.maclib@dm!
00000100 WORDS WRITTEN TO IMAGE
```

Using ncgram append: Example

```
% ncgram append

Enter name of the directory containing text file: my.grammar

Enter name of text file: insurfstate

Enter name of directory containing grm & voc files:
my.grammar

Do you wish to check APT statements: no

Enter name of check file:
Processing text file
Creation complete, beginning correctness check
Correctness check complete, beginning copy
Compilation successful, no errors occurred
The following files have been created
my/grammar/_voc
my/grammar/_grm
```

Using macutil: Example

```
% macutil
           ************************************
          *              MACUTIL               *
          *          CVMAC Macro Utility       *
          *                                    *
          *                                    *
          *************************************

 Enter 'h' for help at any prompt ...
 ------------------------------------



type in the pathname of the required macro or
the directory which includes the required macro file(s)  >*:
macro/insurf

Type in the path name of the append/grammar text file >*:
my/grammar/insurfstate

Type in the directory name for source grammar files >*:
/usr/apl/cadds/data/cam/
ncmill/grammar

Type in the destination directory for language files >*:
my/grammar.
```

Using checkutil: Example

```
% checkutil
CHECKUTIL
Type 'h' to see documentation or <CR> to begin
=>
What directory do you want the check library to be in? (<CR>
for
current.  Start with / if you are giving full path name.)
=>
Using directory /usr2/cadds/andy/check
Enter the check library name you want to use, <CR> for
default (chklib)
=>
```

Please note: For more examples, including examples of seqlist, sequence_output, and nlinker, refer to the *CVNC System User Guide and Menu Reference*.

Appendix C

Pass-through Statements and Macros

Append Text Files

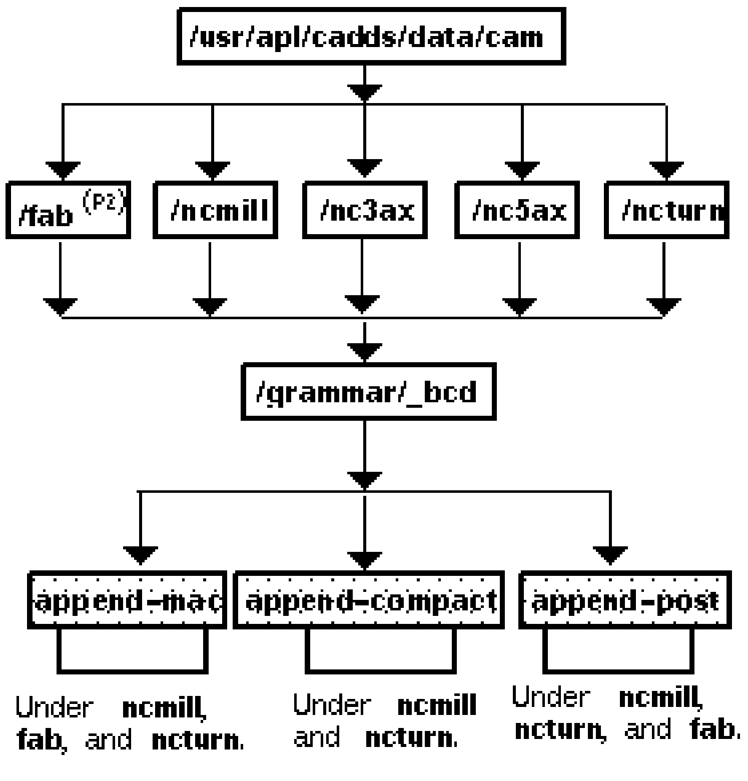
Append text files contain grammar file statements for CVNC-supplied pass-through statements and macros. Append text files can be edited to modify a pass-through statement or macro syntax to meet your site-specific needs.

CVNC provides three append text files.

append-mac	Contains macro references.
append-post	Contains CLFile and APT pass-through statements.
append-compact	Contains COMPACT II pass-through statements.

The following chart illustrates the location of these files:

Figure C-1 Location of Append Text Files



The following sections show the contents of each append text file.

The append-post File

The append-post file contains CLFile and APT pass-through statements. There is an append-post file for CVNC-M2, CVNC-P2, and CVNC-T2.

CVNC-P2 Pass-through Statements

```
<#  DEFINE APT STATEMENTS
<#
%STMT AUXFUN %NEXP $OUTBLK
%STMT BREAK
%STMT END
%STMT INSERT %TEXP
%STMT LEADER %NEXP
%STMT MACHIN %TEXP %MGIVE(%NEXP)
%STMT MODE %CHOOSE ( ABSOL %OR INCR )
%STMT OPSKIP $ONOFF
%STMT OPSTOP
%STMT PARTNO %TEXP
%STMT POSITN %NEXP %MGIVE ( CLEAR %NEXP )
<#                  %MGIVE ( XCOORD %NEXP )
<#                  %MGIVE ( YCOORD %NEXP )
%STMT PPRINT %TEXP
%STMT PREFUN %NEXP $OUTBLK
%STMT RESET
%STMT REWIND
%STMT SEQNO %CHOOSE ( %NEXP %MGIVE (INCR %NEXP)
<#                     %OR AUTO
<#                     %OR ON %OR OFF )
%STMT STOP
%STMT TRANS $THREE
%STMT TMARK %NEXP
%STMT UNLOAD %MGIVE ( %NEXP %NEXP )
<#
<#  *** STATEMENTS FOR CVFAB OUTPUT MACROS ***
<#
<#  DEFINE METASYMBOLS
<#

%FOR $LINEAR %USE LINEAR %NEXP
%FOR $STEP %USE STEP %NEXP
<#
%STMT CYCLE %CHOOSE ($LINEAR %OR $STEP %OR OFF)
%STMT PPUNCH $ONOFF
%STMT TURRET %NEXP DIAMTR %NEXP
%STMT ROTATE CAXIS %NEXP
```

CVNC-M2 Apt Pass-through Statements (append-post)

```
<#   DEFINE APT STATEMENTS
<#
%STMT AUXFUN %NEXP $OUTBLK
%STMT BREAK
%STMT DELAY $TIME
%STMT END
%STMT INSERT %TEXP
%STMT LEADER %NEXP
%STMT MACHIN %TEXP %MGIVE(%NEXP)
%STMT MODE %CHOOSE ( ABSOL %OR INCR )
%STMT OPSKIP $ONOFF
%STMT OPSTOP
%STMT PARTNO %TEXP
%STMT PPRINT %TEXP
%STMT PREFUN %NEXP $OUTBLK
%STMT RESET
%STMT REWIND
%STMT SELCTL %NEXP
%STMT SEQNO %CHOOSE ( %NEXP %MGIVE (INCR %NEXP)
<#                     %OR AUTO
<#                     %OR ON %OR OFF )
%STMT SET HED %CHOOSE ( POSX %OR POSY %OR NEGX %OR NEGY
<#                       %OR NORMAL )
%STMT SPINDL %CHOOSE ( LOCK %OR ORIENT %NEXP )
%STMT STOP

%STMT TRANS $THREE
%STMT TMARK %NEXP
<#
<#   DEFINE METASYMBOLS FOR CYCLE STATEMENTS
<#
%FOR $DWL    %USE %MGIVE ( DWELL %MCHOOSE(%NEXP %OR REV
%NEXP ))
%FOR $FED    %USE %MGIVE( FEDTO %NEXP )
%FOR $FTYPE  %USE %MCHOOSE  ( IPM %NEXP %OR MMPM %NEXP
<#                           %OR IPR %NEXP %OR MMPR %NEXP )
%FOR $RAP    %USE %MGIVE ( RAPTO %NEXP )
%FOR $RET    %USE %MGIVE ( RTRCTO %NEXP )
%FOR $MILL %USE MILL $RAP $FED $RET $FTYPE $DWL
%FOR $TEN    %USE %NEXP %REPEAT (%NEXP) %UPTO 9 %TIMES
%FOR $F10    %USE %MGIVE ( FEDTO $TEN )
%FOR $THRU %USE THRU %MGIVE (RAPTO) $TEN $F10 $RET $FTYPE
$DWL
<#
<#   CYCLE STATEMENTS
<#
%STMT CYCLE %CHOOSE ( $ONOFF %OR $MILL %OR $THRU )
```

CVNC-T2 Aptsource Pass-through Statements (append-post)

```
<# ###  DEFINE APT STATEMENTS ###
<#
%STMT AUXFUN %NEXP $OUTBLK
%STMT BREAK
%STMT DELAY $TIME
%STMT END
%STMT INSERT %TEXP
%STMT LEADER %NEXP
%STMT MACHIN %TEXP %MGIVE(%NEXP)
%STMT MODE %CHOOSE ( ABSOL %OR INCR )
%STMT OPSKIP $ONOFF
%STMT OPSTOP
%STMT PARTNO %TEXP
%STMT PPRINT %TEXP
%STMT PREFUN %NEXP $OUTBLK
%STMT RESET
%STMT REWIND
%STMT SEQNO %CHOOSE (%NEXP %MGIVE ( INCR %NEXP)
<#                            %OR AUTO
<#                            %OR ON %OR OFF )
%STMT SPINDL %CHOOSE ( LOCK %OR ORIENT %NEXP )
%STMT STOP
%STMT TRANS $THREE
%STMT TMARK %NEXP
```

The append-compact File

The append-compact file contains COMPACT II pass-through statements.There is an append-compact file for CVNC-M2 and CVNC-T2.

CVNC-M2 and CVNC-T2 COMPACT II Pass-through Statements (append-compact)

```
<# DEFINE COMPACT II STATEMENTS
<#
%STMT MOVE %CHOOSE (COF %OR CON %OR CON1 %OR CON2 %OR CON3)
<#          %MCHOOSE (BD %OR BDON %OR BDOFF
<#                   %OR MAXRPM %OR MINRPM
<#                   %OR ABSO %OR INCO
<#                   %OR QUIN %OR QUOT
<#                   %OR CAM %OR DRAWBAR %OR LOCK
<#                   %OR SEQ %OR SENSE %OR TOOL %NEXP %OR
RESET )
%STMT END %MCHOOSE (NORWD %OR STOP %OR STOPS %OR TOOL %NEXP
%OR RETS)
%STMT HOME
%STMT IDENT %TEXP
%STMT INIT %CHOOSE (METRIC / IN %MGIVE (INCH / OUT)
<#              %OR  INCH / IN %MGIVE (METRIC / OUT))
%STMT INSRT %TEXP
%STMT MACHIN %TEXP
```

The append-mac File

The append-mac file contains CVNC-supplied macro references.There is an append-mac file for CVNC-M2, CVNC-P2, and CVNC-T2. Each is illustrated below and following pages.

CVNC-M2 append-mac File

```
<#            MACRO GRAMMAR FOR CVNC-M2
<#
<#  CADDS MACROS
<#
%MACRO INSLIN %DOC 'INSLIN' %LIMIT(%XYZLOC) %FROM 2 %UPTO 2
%MACRO INSARC %DOC 'INSARC' RAD %NEXP AGO %NEXP AEND %NEXP
%XYZLOC
%MACRO INSFIL %DOC 'INSFIL' RAD %NEXP %LIMIT (%ENT) %FROM 2
%UPTO 2
<#                           BIAS %LIMIT (%XYZLOC) %FROM 1
%UPTO 1
%MACRO INSCIR %DOC 'INSCIR' RAD %NEXP %LIMIT (%XYZLOC) %FROM
1 %UPTO 1
%MACRO INSPOI %DOC 'INSPOI' %LIMIT(%XYZLOC) %FROM 1 %UPTO 1
%MACRO DELENT %DOC 'DELENT' %LIMIT (%ENT) %FROM 1 %UPTO 100
%MACRO VERENT %DOC 'VERENT' %ENT
<#
<#  NC MACROS
<#
%MACRO TLCHG %DOC 'TLCHG' %NEXP %TEXP SELCTL %NEXP
<#
%MACRO LEGROUP %DOC 'LEGROUP' %LIMIT (%ENT) %FROM 1 %UPTO
100
<#
%MACRO SPROF %DOC 'SPROF' %TEXP %CHOOSE (OPEN %OR CLOSED)
<#           %CHOOSE (ROLL %OR NOROLL) ISTK %NEXP RSTK %NEXP
<#
%VMACRO DPROF %DOC 'DPROF'  %TEXP %ARG 1
<#                %CHOOSE (OPEN %ARG 2 %OR CLOSED %ARG 2)
<#                %CHOOSE (ROLL %ARG 3 %OR NOROLL %ARG 3)
<#                ZSTEP %NEXP %ARG 4 ZEND %NEXP %ARG 5
                %MGIVE (ANGLE %NEXP %ARG 6)
<#
%VMACRO DPOCK %DOC 'DPOCK' %TEXP %ARG 1
<#           %MGIVE (RTOL %NEXP %ARG 2)
<#           ZSTEP %NEXP %ARG 3
<#           ZEND %NEXP %ARG 4
<#           %MGIVE (MAXSTEP %NEXP %ARG 5)
<#          %MGIVE (APPROX %ARG 6) %MGIVE (ANGLE %NEXP %ARG
7)
```

```
<#           %MGIVE (ISLAND %TEXP %ARG 8 %MGIVE (%TEXP %ARG
9
<#            %MGIVE (%TEXP %ARG 10 %MGIVE (%TEXP %ARG 11
<#            %MGIVE (%TEXP %ARG 12 %MGIVE (%TEXP %ARG 13
<#            %MGIVE (%TEXP %ARG 14 %MGIVE (%TEXP %ARG
15))))))))
```

CVNC-T2 append-mac File

```
<#
<#            MACRO GRAMMAR FOR CVNC-T2
<#
<#  CADDS MACROS
<#
%MACRO INSLIN %DOC 'INSLIN' %LIMIT(%XYZLOC) %FROM 2 %UPTO 2
%MACRO INSARC %DOC 'INSARC' RAD %NEXP AGO %NEXP AEND %NEXP
%XYZLOC
%MACRO INSFIL %DOC 'INSFIL' RAD %NEXP %LIMIT (%ENT) %FROM 2
%UPTO 2
<#                 BIAS %LIMIT (%XYZLOC) %FROM 1 %UPTO 1
%MACRO INSCIR %DOC 'INSCIR' RAD %NEXP %LIMIT (%XYZLOC) %FROM
1 %UPTO 1
%MACRO INSPOI %DOC 'INSPOI' %LIMIT(%XYZLOC) %FROM 1 %UPTO 1
%MACRO DELENT %DOC 'DELENT' %LIMIT (%ENT) %FROM 1 %UPTO 100
%MACRO VERENT %DOC 'VERENT' %ENT
<#
<#  NC MACROS
<#
%MACRO TLCHG %DOC 'TLCHG' %NEXP %TEXP
<#
%MACRO LEGROUP %DOC 'LEGROUP' %LIMIT (%ENT) %FROM 1 %UPTO 10
<#
%VMACRO FTURN %DOC 'FTURN' %TEXP %ARG 1 MATL %TEXP %ARG 2
<#
>                 MAXDEPTH %NEXP %ARG 3
<#
>                      SAFDIST %NEXP %ARG 4  %MGIVE (UDEPTH
%NEXP %ARG 5) <#
>                        RSTOCK %NEXP %ARG 6 FINFEED %NEXP
%ARG 7           <#
>                        XYSIDE %XYZLOC %ARG 8
%MACRO RFACE %DOC 'RFACE' %XYZLOC ENDLOC %XYZLOC FINLOC
<#                        %XYZLOC MAXDEPTH %NEXP RSTOCK
<#                        %NEXP

<#
%VMACRO GROOVE %LIMIT(%ENT) %FROM 3 %UPTO 3 %ARG 1
<#                        CLEAR %NEXP %ARG 2
<#                        TLWIDTH %NEXP %ARG 3
<#                      %MCHOOSE (SWEEP %OR NOSWEEP) %ARG 4
<#                        %MCHOOSE (BREAK %ARG 5 RAD %NEXP
%ARG 6
<#                        %OR NOBREAK %ARG 5)
<#                      %MCHOOSE (DWELL %OR NODWELL) %ARG 7
<#
%MACRO BLEND %DOC 'BLEND' RAD %NEXP %LIMIT (%ENT) %FROM 2
%UPTO 255
<#                        %LIMIT (%XYZLOC) %FROM 1 %UPTO 1
```

CVNC-P2 append-mac File

```
<#
<#            MACRO GRAMMAR FOR CVNC-P2
<#
<#  CADDS MACROS
<#
%MACRO INSLIN %DOC 'INSLIN' %LIMIT(%XYZLOC) %FROM 2 %UPTO 2
%MACRO INSARC %DOC 'INSARC' RAD %NEXP AGO %NEXP AEND %NEXP
%XYZLOC
%MACRO INSFIL %DOC 'INSFIL' RAD %NEXP %LIMIT (%ENT) %FROM 2
%UPTO 2
<#                           BIAS %LIMIT (%XYZLOC) %FROM 1
%UPTO 1
%MACRO INSCIR %DOC 'INSCIR' RAD %NEXP %LIMIT (%XYZLOC) %FROM
1 %UPTO 1
%MACRO INSPOI %DOC 'INSPOI' %LIMIT(%XYZLOC) %FROM 1 %UPTO 1
%MACRO DELENT %DOC 'DELENT' %LIMIT (%ENT) %FROM 1 %UPTO 100
%MACRO VERENT %DOC 'VERENT' %ENT
<#
<#  NC MACROS
<#
%MACRO LEGROUP %DOC 'LEGROUP' %LIMIT (%ENT) %FROM 1 %UPTO
100
<#
%MACRO GRID  %DOC 'GRID' %TEXP %CHOOSE (XMAX %OR XNUM) %NEXP
<#                 %CHOOSE (YMAX %OR YNUM) %NEXP
<#                 LOWLEFT %LIMIT (%XYZLOC) %FROM 1 %UPTO 1
<#                 UPRIGHT %LIMIT (%XYZLOC) %FROM 1 %UPTO 1
<#                 VERPUNWD %NEXP HORPUNWD %NEXP
```

APT/CLFile Word Tables

These tables list the APT/CLFile major words, minor words, and synonyms for major and minor words that are supported by CVNC.

The ncgram append command checks the words in any statements you enter in the grammar file to see if they are valid APT/CLFile words. This is done by referencing the following files:

```
/usr/apl/cadds/data/cam/aptwords/_bcd/major
```

```
/usr/apl/cadds/data/cam/aptwords/_bcd/minor
```

```
/usr/apl/cadds/data/numcon/apt/synonyms
```

The synonyms file contains synonyms for some major and minor words. You can define a pass-through statement using these synonyms instead of the actual APT/CLFile words.

Major Words

Table C-1 Major APT/CLFile Words

Word	Integer Code
AIR	1011
ARCSLP	1029
ASLOPE	1053
AUXFUN	1022
BREAK	16
CAMERA	1047
CHECK	1023
CHUCK	1073
CIRCLE	9015
CLAMP	1074
CLDIST	1071
CLEARP	1004
CLRSRF	1057
COOLNT	1030
CORNED	1088
COUPLE	1049
CUTCOM	1007
CUTTER	6000
CYCLE	1054
DELAY	1010
DISPLY	1021
DRAFT	1059
DRAWLI	23
DRESS	8
DWELL	1058
END	1
FACEML	22
FEDRAT	1009
FROM	9001
GOHOME	17
GOTO	9002
HEAD	1002
IFRO	1032
INDEX	1039
INSERT	1046

Word	Integer Code
INTCOD	1020
ISTOP	4
LATHE	1100
LEADER	1013
LETTER	1043
LINTOL	1067
LOADTL	1055
LOCKX	21
LPRINT	1065
REWIND	1006
ROTABL	1026
ROTATE	1066
ROTHED	1035
SAFETY	1028
SAFPOS	1094
SELCTL	1056
SELECT	1101
SEQNO	1019
SETUP	1086
SLOWDN	1063
SPINDL	1031
STAN	1080
STOP	2
SWITCH	6
THREAD	1036
TMARK	1005
TOOLNO	1025
TRACUT	1038
TRANS	1037
TURRET	1033
ULOCKX	20
UNLOAD	10
VTLAXS	1070
ZERO	13
3PT2SL	43
4PT1SL	44
5PT	45
AAXIS	140

Word	Integer Code
ABSOL	497
ADJUST	159
ALL	51
ANTSPI	176
ARC	182
AT	189
ATANGL	1
AUTO	88
AVOID	187
BACK	317
BAXIS	141
BCD	165
BINARY	164

Minor Words

Table C-2 Minor APT/CLFile Words

Word	Integer Code
BLACK	130
BLUE	133
BORE	82
BOTH	83
BRKCHP	288
CAM	237
CAXIS	142
CCLW	59
CENTER	2
CHUCK	138
CIRCUL	75
CLW	60
COARSE	195
COLLET	139
CONST	64
CROSS	3
CSINK	256
CSS	319
CTRLIN	126
CUTANG	160
CUTS	511
DARK	137
DASH	124
DECR	65
DEEP	153
DELZ	320
DEPTH	510
DIAMET	509
DIAMTR	205
DITTO	127
DOTTED	125
DOWN	113
DRAG	278
DRILL	163
DWELL	279

Word	Integer Code
ENDARC	58
FACE	81
FEDTO	281
FEET	304
FINCUT	512
FINE	193
FLOOD	89
FOURP	T101
FPM	322
FPR	323
FRONT	148
FULL	147
FUNOFY	4
GAGE	324
GAPLES	180
GLATHE	217
GLPOST	216
GPPOST	218
GREEN	132
HEIGHT	325
HIGH	62
HOLDER	157
HOLDIA	293
IN	48
INCHES	303
INCR	66
INHIBT	197
INTENS	134
INTERC	46
INTOF	5
INVERS	6
IPM	73
IPR	74
LARGE	7
LAST	52
LEAD	326
LEFT	8
LENGTH	9

Word	Integer Code
LIGHT	100
LINCIR	95
LINEAR	76
LITE	135
LOCK	114
LOW	63
LOWLFT	327
MAIN	513
MANOP	328
MANUAL	158
MAXIPM	96
MAXRPM	79
MED	136
MEDIUM	61
MILL	151
MINUS	10
MIRROR	56
MIST	90
MM	301
MMPM	315
MMPR	316
MODIFY	55
MULTRD	119
MXMMPM	506
MXPERM	507
NCPOST	219
NCPRES	211
NEGX	11
NEGY	12
NEGZ	13
NEUTRL	166
NEXT	162
NIXIE	99
NOBACK	194
NODRAG	289
NOMORE	53
NOREVR	329
NORMAL	111

Word	Integer Code
NORMDS	87
NOW	161
NOX	14
NOY	15
NOZ	16
OFF	72
OMIT	186
ON	71
OPEN	50
OPTION	144
ORIENT	246
OSETNO	508
OUT	49
OXYGEN	169
PALLET	239
PARAB	77
PARLEL	17
PART	178
PAS	T70
PEN	128
PERMIN	501
PERPTO	18
PERREV	504
PERSP	67
PLANE	3003
PLASMA	499
PLUS	19
POSX	20
POSY	21
POSZ	22
PREHET	171
PSTAN	146
PTNORM	104
PTSLOP	103
QUILL	287
RADIUS	23
RAIL	93
RAM	500

Word	Integer Code
RANDOM	174
RANGE	145
RAPOUT	330
RAPTO	280
REAM	262
REAR	149
RED	131
RETAIN	184
REV	97
RIGHT	24
ROTREF	68
RPM	78
RTHETA	106
RTRCTO	295
SADDLE	150
SAME	54
SCALE	25
SCRIBE	129
SETANG	156
SETOOL	155
SFM	115
SIDE	94
SLOPE	47
SMALL	26
SMM	505
SOLID	123
SPLINE	105
START	57
STEP	92
STEPR	332
SYMBOL	235
TABLE	177
TANON	109
TANTO	27
TAP	168
TAPER	540
TAPKUL	91
THETAR	107

Word	Integer Code
THRU	152
TIMES	28
TLANGL	294
TO	69
TOOL	3017
TORCH	172
TPI	143
TRANSL	29
TRAV	154
TRFORM	110
TUL	240
TWOPT	102
TYPE	98
UAXIS	227
UNIT	30
UP	112
UPRGT	335
VAXIS	228
WAXIS	229
XAXIS	84
XCOORD	116
XLARGE	31
XOFF	290
XSMALL	32
XYPLAN	33
XYROT	34
XYVIEW	120
XYZ	108
YAXIS	85
YCOORD	117
YLARGE	35
YOFF	291
YSMALL	36
YZPLAN	37
YZROT	38
YZVIEW	121
ZAXIS	86
ZCOORD	118

Word	Integer Code
ZIGZAG	185
ZLARGE	39
ZSMALL	40
ZXPLAN	41
ZXROT	42
ZXVIEW	122

Synonyms

Table C-3 APT/CLFile Word Synonyms

Synonym	Word
A	ATANGL
AV	AVOID
CT	CENTER
C	CIRCLE
CO	COOLNT
CTR	CUTTER
CY	CYCLE
CYL	CYLNDR
DR	DRILL
F	FEDRAT
GB	GOBACK
GD	GODLTA
GDN	GODWN
GF	GOFWD
GH	GOHOME
GL	GOLFT
GR	GORGT
GT	GOTO
GU	GOUP
HD	HEAD
IP	INDIRP
IV	INDIRV
I	INTOF
LG	LARGE
LT	LEFT
LIN	LINE

Synonym	Word
LINR	LINEAR
LDT	LOADTL
M	MACRO
MX	MATRIX
MY	MODIFY
NM	NOMORE
OPT	OPTION
PLL	PARLEL
PA	PATERN
PP	PERPTO
PL	PLANE
PKT	POCKET
P	POINT
PPRN	PPRINT
R	RADIUS
RAP	RAPID
RE	REAM
REF	REFSYS
REM	REMARK
RES	RESERV
RET	RETRCT
RT	RIGHT
ROT	ROTABL
SEL	SELECT
STL	SELCTL
SEQ	SEQNO
SM	SMALL
SP	SPINDL
TAB	TABCY
TBL	TABLE
TT	TANTO
THD	THREAD
TL	TLLFT
TN	TLON
TR	TLRGT
TRT	TURRET
V	VECTOR
XL	XLARGE

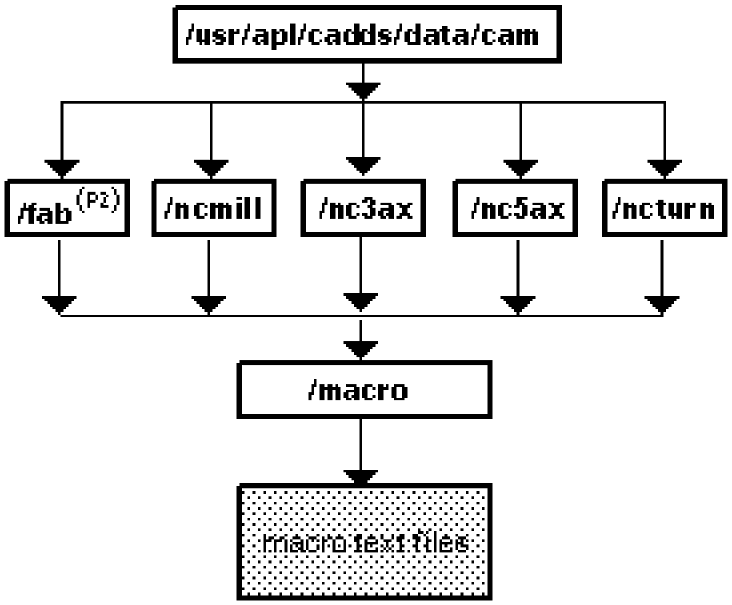
Synonym	Word
XS	XSMALL
YL	YLARGE
YS	YSMALL
ZL	ZLARGE
ZS	ZSMALL

CVNC-supplied Macro Text Files

CVNC-supplied macro text files contain CVMAC source code.

The following chart illustrates the location of these files. For each macro, there is a source code text file, a code image file, and a data image file. These last two files are created when the macro is compiled.

Figure C-2 Location of Macro Text File



The following pages show the macro text files for the CVNC-supplied TLCHG macro.

TLCHG Macro

```
PROC TLCHG (PODNO,&TNAM,&SELCTL,SELCTL)
<#
<#   CVMAC MACRO TO AUTOMATE TOOL CHANGE OPERATIONS
<#
<#   PURPOSE:
<#        ALLOWS OPERATOR TO AUTOMATE TOOL CHANGE OPERATIONS
<#        BY DOING
<#            CHANGE TOOL COMMAND
<#            SELECTING OF NEXT TOOL
<#            CLEARS TOOL TO HOME-PT PRIOR TO TOOL-CHANGE
<#            TURNS SPINDLE OFF
<#            SETS TOOL-OFFSET REGISTERS
<#   COMMAND SYNTAX
<#       %MACRO TLCHG %NEXP %TEXP SELCTL %NEXP
<#
<#
DECLARE DOUBLE PODNO,SELCTL
DECLARE TEXT &TNAM,&SELCTL
        !COOLANT OFF
        !CLEAR HOME
        !SPEED OFF
        !CHGTOOL {PODNO} '{&TNAM}' LENREG {PODNO} DIAREG
{PODNO}
        !SELCTL {SELCTL}
RETURN
```

Index