THE VWS_CADM USER INTERFACE OF THE VERTICAL WORKSTATION OF THE AUTOMATED MANUFACTURING RESEARCH FACILITY AT THE NATIONAL BUREAU OF STANDARDS

I. INTRODUCTION

1. CONTENTS

This paper discusses the vws_cadm user interface for the Vertical Workstation (VWS) of the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards. The descriptions pertain to the system in use during the summer of 1987. The vws_cadm user interface provides a user-friendly means of using three of the major VWS modules and a variety of input and output capabilities of the VWS

There are two appendices to this paper. The first is a users manual for vws_cadm and the second is a users manual for the VWS Part Design Editor.

2. AUDIENCE

The paper is intended to be useful to people interested in concepts and technical details of the VWS, particularly AMRF personnel who are running the VWS or maintaining or improving the software for the VWS. The paper is intended to be useful also to other researchers in automated manufacturing. A knowledge of the computer language LISP is useful but not essential to reading this paper. Detailed documentation of the LISP functions that are involved with the systems described here is being prepared separately.

3. BRIEF VWS DESCRIPTION

The VWS is a computer-integrated automated machining workstation. It includes a control system, a computer-aided design system, an automatic process planning system, and an automatic NC-code generator. The principal machinery is a vertical machining center (Monarch VMC-75 with a GE2000 controller) and a robot (Unimate 4070 with a Val II controller) to tend the milling center. There is quite a bit of ancillary hardware. The system is controlled from a microcomputer (Sun 3/160 with 6Mb memory, BW monitor). Running in stand-alone mode, it is possible to design and machine a simple metal part within an hour. The VWS may also be run as an integrated part of the AMRF. The workstation is described in more detail in [K&J1].

The software for the VWS is written in Franz LISP dialect of the computer language LISP. In this paper this software is called the VWS2 system. Six principal modules comprise the VWS2 system: the Production Management Operating System (the control system), the State Table Editor, the Equipment Program Generator, the Part Design Editor, the Process Planner, and the Data Execution module.

The Part Design Editor, Process Planning and Data Execution modules, as well as other system

CONTENTS

Page

I. INTRODUCTION 1
1. CONTENTS 1
2. AUDIENCE 1
3. BRIEF VWS DESCRIPTION 1
4. RELATED READING 2

II. VWS_CADM 3
1. OVERVIEW 3
2. TOP LEVEL OPTIONS 4
3. PROCESS PLANNING OPTION 4
4. DATA EXECUTION OPTION 5
5. INPUT OPTION 6
5.1. Introduction 6
5.2. Load Design 6
5.3. Load LISP-readable Process Plan 6
5.4. Read in AMRF Standard Format Process Plan 6
5.5. Load a LISP-readable Workpiece Description 7
6. OUTPUT OPTION 7
6.1. Introduction 7
6.2. Print a LISP-readable Process Plan 7
6.3. Print an AMRF Standard Format Process Plan 7
6.4. Print a LISP-readable Design 8
6.5. Print a Human-readable Design 8
6.6. Draw a Design 8
6.7. Draw a Workpiece 8
6.8. Draw Tool Path 8
6.9. Erase the Existing Picture 8
7. VWS_CADM SOFTWARE 9

REFERENCES 10

APPENDIX A. VWS_CADM USERS MANUAL

APPENDIX B. DESIGN EDITOR USERS MANUAL

capabilities, may be accessed by the user through a small user interface called vws_cadm. Vws_cadm asks the user questions about what the user wants to do and then activates the appropriate module or other capability accordingly.

To produce a part from scratch, the user sits at the Sun workstation and creates a design using the Part Design Editor. The Process Planner is then called to write a plan for how to machine a part of that design. Next NC-code is generated automatically from the design and the plan by the Data Execution module. Finally the user tells the control system to make the part. The control system coordinates the activities of the workstation equipment so that the part blank is loaded onto the milling machine, the NC-code is sent to the milling machine and executed (making the part), and the finished part is unloaded.

To make a part using existing data, the VWS controller calls on the local database manager to retrieve several types of data: tray contents, VWS process plan, and NC-code. Following the process plan, the VWS controller directs workstation activity so that the robot loads the part onto the milling machine, the NC-code is sent to the milling machine and executed, and the finished part is unloaded.

4. RELATED READING

This paper is one of about a dozen papers being prepared as part of the AMRF documentation to describe all aspects of the VWS. The others are [JUN], [KRA1], [KRA3], [KRA4], [K&J2], [K&S2], [KR&W], [LOVE], [NA&J], and [RUDD]. Other papers, prepared for professional meetings, also describe the VWS [KRA2], [K&J1], and [K&S1].

II. VWS_CADM

1. OVERVIEW

The vws_cadm user interface provides easy access to:
A. the Design Editor module,
B. the Process Planning module,
C. the Data Execution module,
D. four VWS2 input capabilities, and
E. eight VWS2 output capabilities.

The user engages in a dialog with vws_cadm by giving commands and answering questions. All input to vws_cadm is through the keyboard and is directed to the window on the Sun in which the VWS2 system is running.

The vws_cadm prompt is an arrow that looks like this ==> . Before the prompt is displayed, vws_cadm always lists or describes the choices the user has inside angle brackets < > .

The choices inside the angle brackets are given very tersely, usually by one-letter or two-letter abbreviations. However, vws_cadm includes a help facility that is available whenever the vws_cadm prompt is waiting for user input. The user simply types "h" for help, and vws_cadm explains the meaning of the choices in more detail.

Three other options in addition to help are available whenever the vws_cadm prompt is showing: to break into LISP, to quit, or to redo the last item.

Vws_cadm gives the user access to several other interactive programs which do not have these options. The user can tell when vws_cadm is speaking by the appearance of the prompt. These other programs have different prompt symbols.

Vws_cadm was built to make the VWS2 system easy to use for knowledgeable users. It is not intended to give novices an easy introduction to the system. First-time users may be confused by the switch from vws_cadm to the other interactive programs accessible from vws_cadm, since each of the others has its own user interface routines.

Vws_cadm gives access to VWS2 capabilities only. There are several data preparation and handling capabilities that a user might want to have that are currently best handled by UNIX or the text editor, EMACS. To use these capabilities, the user should select a window other than the one in which the VWS2 system is running.

UNIX is convenient for:
A. Displaying a design, process plan, or NC-code file.
B. Moving a file from one directory to another.
C. Deleting a design, process plan, or NC-code file.
D. Printing a design, process plan, or NC-code file.
E. Making a picture by doing a screen dump to a laser printer.
EMACS may be used by a knowledgeable user for:

A. Editing a process plan or NC-code file (but not a design file).

There are many VWS2 system capabilities that are not available through vws_cadm. The control system built by Dr. J. Jun [JUN] provides a large number of capabilities for operating the workstation and for dealing with the AMRF or local VWS database. The control system has its own user interface.
The VWS2 State Table Editor module built by Dr. Jun is called directly from LISP.

2. TOP LEVEL OPTIONS

At the top level of vws_cadm, the user is presented with the five options A-E shown at the beginning of the last section.

If the Design Editor module option is chosen, vws_cadm starts the Design Editor up, and the user deals with the Editor until quitting the editor. Then the five vws_cadm options are presented to the user again. The user is referred to Appendix B of this paper and to [K&J2] for more information about the design protocol and the Design Editor.

If one of the other four options is chosen, another level of vws_cadm is entered, and the user must answer a set of questions regarding what the user wants to do. In the second level, the user may always back up and redo the previous item by entering "r", or pop back to the top level by entering "q" instead of answering the question.

3. PROCESS PLANNING OPTION

To use the Process Planning module, the user must specify:
A. the name of design for which a plan is to be made,
B. the material from which the part is to be made,
C. whether the four parameters: spindle speed, feed rate, pass depth and stepover, should be included in those steps of the process plan in which they are appropriate,
D. what the plan_id should be in the VWS2 LISP environment,
E. whether the plan should be printed in a file in LISP-readable form, and, if so, what the name of the file should be, and
F. whether the plan should be printed in a file in standard AMRF form, and, if so, what the name of the file should be.

Once the user has answered these questions, vws_cadm runs the Process Planning module accordingly. When the process plan is complete or when it aborts, the user is notified, and vws_cadm pops back to the top level. If the Process Planning module aborts, the user is notified of the reason. The most common reason for failure is that no tool can be found in the tool catalog for making some feature or subfeature.

Any files that the user has asked to have prepared are stored in the directory from which the VWS2 system is running.

When the user gives the design name, vws_cadm checks that the design is a valid design. If it is not a valid design, the user is prompted to enter another design name.

If the user specifies any material other than aluminum, brass, steel, or monel, the user is prompted again to enter a material.

If the plan_id given by the user is already being used, the user is prompted for another plan_id.

If the user has specified a file name for printing the process plan, and a file with that name already exists, the user is prompted to enter a different file name.

The user is referred to [KRA1] and [KRA2] for more information about the Process Planning module.

4. DATA EXECUTION OPTION

To use the Data Execution module, the user must answer questions to specify:

A. the name of the plan to be executed,
B. the name of the workpiece to be used for plan execution,
C. whether to mill the part in the vise or on a pallet,
D. whether to establish z-zero by measuring up from the bottom of the fixture or down from the top of the part,
E. what the level of verification should be,
F. whether NC-code should be written, and if so, what the name of the NC-code file should be,
G. whether a drawing should be made to emulate the machining process,
H. whether the enhanced process plan should be saved, and if so, what the name of the file should be, and
I. whether the description of the finished workpiece should be saved, and if so, what the name of the file should be.

Once the user has answered these questions, vws_cadm runs the Data Execution module accordingly. The Data Execution module is rather verbose and sends the user many messages about the progress it is making. When the module is finished or when it aborts, the user is notified, and vws_cadm pops back to the top level. If the module aborts, the user is notified of the reason. When the module is run with verification on (which it almost always should be), up to several hundred verification rules may be applied. A violation of any rule will stop the module.

Any files that the user has asked to have prepared are stored in the directory from which the VWS2 system is running.

When the user gives the plan name, vws_cadm checks that the plan is a valid plan. If it is not valid, the user is prompted to enter another plan name.

When the user gives the workpiece name, vws_cadm checks whether a description of the workpiece already exists in the LISP environment. If it already exists, it is checked to see that it is a valid workpiece description and that the workpiece is made of the material specified in the plan. If the check fails, the user is notified of the reason and prompted for another workpiece name. If there is no description of the workpiece in the LISP environment, vws_cadm makes one up, using the given workpiece name, the material specified in the plan, and the block size given in the design. This workpiece description is handed to the Data Execution module for its use.

In answering the questions A - I above, if the user has specified a file name, and a file with that name already exists, the user is prompted to enter a different file name.

The user is referred to [KR&W] for more information about the Data Execution module.

5. INPUT OPTION

5.1. Introduction

If the user chooses the input option at the top level of vws_cadm, four input options are offered at the second level:
A. Load a LISP-readable design file from the design directory.
B. Load a LISP-readable process plan file.
C. Read in an AMRF standard format process plan file.
D. Load a LISP-readable workpiece description file.

5.2. Load Design

The user is prompted to enter the name of a design. Vws_cadm looks in the design subdirectory of the directory in which the VWS2 system is operating for a file of the given name with the suffix ".pd". If the file is found it is loaded. In addition, the name of the design is placed on the Design Editor module's PART_DESIGN_LIST so that the design may be edited without having to load it again.

If the design is not found, vws_cadm prints an error message on the screen.

In either case the user is prompted again to choose an input option.

5.3 Load LISP-readable Process Plan

The user is prompted to enter the complete path name of a LISP-readable process plan file. If the file is found it is loaded. If not, an error message is printed. In either case the user is prompted again to choose an input option.

5.4. Read in AMRF Standard Format Process Plan

The user is prompted to enter the complete path name of an AMRF standard format process plan file. If the file is found, the user is also prompted for a name to use as the plan_id (this should normally be the same as the plan_id given in the file, changed to lower case letters). Then the VWS2 standard process plan reading facility is called to read in the process plan, which is set up as the property list of the plan_id given by the user. It may take 30 seconds or more to do the reading.

If the file named by the user is not found, an error message is printed.

In either case the user is prompted again to choose an input option.

5.5. Load a LISP-readable Workpiece Description

The user is prompted to enter the complete path name of a LISP-readable workpiece file. If the file is found it is loaded. If not, an error message is printed. In either case the user is prompted again to choose an input option.

6. OUTPUT OPTION

6.1. Introduction

If the user chooses the output option at the top level of vws_cadm, eight output options are offered at the second level:
A. Print a LISP-readable process plan file.
B. Print an AMRF standard format process plan file.
C. Print a LISP-readable design file.
D. Print a human-readable design file.
E. Draw a design.
F. Draw a workpiece.
G. Draw the tool path.
H. Erase the existing picture.

Regardless of what the chosen option does, when it is finished, the user is prompted again to choose an output option.

6.2. Print a LISP-readable Process Plan

The user is prompted for the name of a process plan. If the given name is not the name of a process plan, an error message is printed.

If the given name is the name of a process plan, the user is prompted for the full UNIX path name of a file in which to print the plan. If the file already exists, the user is asked if it may be overwritten. If overwriting is OK, or if there is no existing file by that name, the plan is written into the file.

6.3. Print an AMRF Standard Format Process Plan

This is just like printing a LISP-readable plan, except for the print format.

6.4. Print a LISP-readable Design

The user is prompted for the name of a design. If the named design is not a proper design, an error message is sent.

If the design is OK, the user is prompted for the name of a file in the "design" subdirectory into which the design should be printed. The name of the file will be the name given by the user with the suffix ".pd" added. If the file already exists, the user is asked if it may be overwritten. If overwriting is OK, or if there is no existing file by that name, the design is written into the file.

6.5. Print a Human-readable Design

This is just like printing a LISP-readable design, except for the print format.

6.6. Draw a Design

The user is prompted for the name of a design. The design must be present in the LISP environment. If the design is on file but not in the environment, it must be loaded before this facility is used. If the named design is not a proper design, an error message is sent.

If the design is OK, the user is prompted to choose a verification level. Then any existing picture is wiped out, and the design is drawn. If verification is on, the design is verified as it is drawn.

6.7. Draw a Workpiece

This is exactly like drawing a design, except that the name of a workpiece is used.

6.8. Draw Tool Path

This option works only if there is a picture in the graphics window which was drawn by the VWS2 Data Execution Module while NC-code was being written. In this case, the tool path specified in the code is drawn on the existing picture. In any other case, an error message is printed.

If the tool path is drawable, the user is asked whether the drawer should stop between cuts or draw slowly.

Tool path drawing is discussed in more detail in Chapter XII of [K&S2].

6.9. Erase the Existing Picture

If there is an existing picture it is erased. If there is no existing picture, nothing happens.

7. VWS_CADM SOFTWARE

The vws_cadm software consists of only seven functions. Six are in the cadm2 subdirectory of the vws2 directory, and one, named "friendly", is in the misc2 subdirectory. Of course, vws_cadm makes use of hundreds of other functions to execute the options available to the user.

The workhorse of the vws_cadm is the "friendly" function. This is a general purpose data-driven shell for conducting dialogs. The arguments to "friendly" are:

A. A message to print at the outset.
B. A disembodied property list of prompt messages.
C. A disembodied property list of help messages.
D. A disembodied property list of LISP code segments to be used as tests.
E. A disembodied property list of error messages.
F. A list which is a control structure for the dialog.

Most of the other six functions is taken up with specifying the lists needed by "friendly".

REFERENCES

[JUN]
Jun, Jau-Shi; "The Vertical Machining Workstation Systems"; to be published as an NBSIR; 1988.

[KRA1]
Kramer, Thomas R.; "Process Plan Expression, Generation, and Enhancement for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 87-3678; National Bureau of Standards; 1987; 56 pages.

[KRA2]
Kramer, Thomas R.; "Process Planning for a Milling Machine from a Feature-Based Design"; Proceedings of Manufacturing International Meeting; Atlanta, Georgia; April 1988; ASME; 1988; Vol. III, pp. 179 -189.

[KRA3]
Kramer, Thomas R.; "The Graphics Subsystem of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; to be published as an NBSIR; 1988; 27 pages.

[KRA4]
Kramer, Thomas R.; "Data Handling in the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3763; National Bureau of Standards; 1988; 62 pages.

[K&J1]
Kramer, Thomas R.; and Jun, Jau-Shi; "Software for an Automated Machining Workstation"; Proceedings of the 1986 International Machine Tool Technical Conference; September 1986; Chicago, Illinois; National Machine Tool Builders Association; 1986; pp. 12-9 through 12-44.

[K&J2]
Kramer, Thomas R.; and Jun, Jau-Shi; "The Design Protocol, Part Design Editor, and Geometry Library of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3717; National Bureau of Standards; 1988; 101 pages.

[K&S1]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention in Data Preparation for a Numerically Controlled Milling Machine"; Proceedings of 1987 ASME Annual Meeting; ASME; 1987; PED-Vol. 25, pp. 195 - 213.

[K&S2]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention and Detection in Data Preparation for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 87-3677; National Bureau of Standards; 1987; 61 pages.

[KR&W]
Kramer, Thomas R.; and Weaver, Rebecca E.; "The Data Execution Module of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3704; National Bureau of Standards; 1988; 58 pages.

[LOVE]
Lovett, Denver; "Equipment Controllers of the Vertical Workstation"; NBSIR 88-3769; National Bureau of Standards; 1988.

[NA&J]
Nakpalohpo, Ibrahim; and Jun, Jau-Shi; "Automated Equipment Program Generator and Execution System of the AMRF Vertical Workstation"; not yet published; 1987; 17 pages.

[RUDD]
Rudder, Frederick; (a paper in preparation describing the VWS hardware and software for the HP-9000 workstation supervisor); to be published as an NBSIR; 1988.

APPENDIX A

VWS_CADM USERS MANUAL

Dr. Thomas R. Kramer
Guest Worker, National Bureau of Standards, &
Research Associate, Catholic University

March 10, 1988

CONTENTS

Page

I. INTRODUCTION A-1
  1. CONTENTS A-1
  2. AUDIENCE A-1
  3. STARTING VWS_CADM A-2
  4. USE OF FONTS IN THIS MANUAL A-3
  5. VWS_CADM OPTIONS A-4
    5.1. Introduction A-4
    5.2. Getting Help from vws_cadm A-4
    5.3. Quitting A-5
    5.4. Redoing the Last Item A-6
    5.5. Breaking into LISP A-7
  6. BUILDING VWS2_LISP A-8
  7. OTHER VWS2 CAPABILITIES A-8
  8. FORMAT OF COMMANDS SECTION A-9

II. COMMANDS A-10
  1. e (Use the Part Design Editor) A-10
  2. p (Make a Process Plan) A-11
  3. x (Use the Data Execution Module) A-13
  4. i (Use Input Capabilities) A-15
    4.1. dl (Load a LISP-readable Design) A-15
    4.2. pl (Load a LISP-readable Process Plan) A-16
    4.3. pf (Read in an AMRF Standard Format Process Plan) A-17
    4.4. wl (Load a LISP-readable Workpiece Description) A-18
  5. o (Use Output Capabilities) A-19
    5.1. pl (Print a LISP-readable Process Plan File) A-19
    5.2. pf (Print an AMRF Standard Format Process Plan File) A-20
    5.3. dl (Print a LISP-readable Design File) A-21
    5.4. dh (Print a Human-readable Design File) A-22
    5.5. gd (Draw a Design) A-23
    5.6. gw (Draw a Workpiece) A-24
    5.7. gt (Draw Tool Path on Existing Picture) A-26
    5.8. ge (Erase Existing Picture) A-28

III. VWS_CADM TUTORIAL A-29

REFERENCES A-38

I. **INTRODUCTION**

1. CONTENTS

This is a users manual for the vws_cadm user interface of the VWS2 software for the AMRF Vertical Workstation (VWS). It has two further sections: a guide to vws_cadm commands, and a tutorial demonstration of vws_cadm. You should try the tutorial after you read the introduction.

2. AUDIENCE

This manual is intended to be used by anyone learning to use vws_cadm or using vws_cadm. A person who is learning vws_cadm should first read the paper to which this is an appendix. That paper gives a general description of vws_cadm which is not repeated here.

Before trying to learn vws_cadm, a person should be familiar with the basics of UNIX and the basics of the window system "suntools" which is part of the standard software for a Sun computer. Both are easy systems to learn. A good manual to use to learn enough about UNIX is Getting Started with UNIX: Beginner's Guide [SUN1]. A good manual to use to learn enough about the window system is Windows and Window Based Tools: Beginners Guide [SUN2].

To use vws_cadm, It is useful but not necessary to know the computer language LISP, since the entire VWS2 system is written in LISP.

3. STARTING VWS_CADM

Vws_cadm should be run on a Sun with at least 4Mb of on-board memory. Any graphics generated while vws_cadm is being used will run slowly unless there are at least 6Mb of on-board memory.

Vws_cadm runs in a suntools window environment. The screen should be set up with at least three windows:
A. A Console window to intercept messages from outside.
B. A vws_cadm command window in the lower left of the screen.
C. A window covering the right-hand third of the screen from the top to near the bottom. This will be used by the VWS Part Design Editor module if it is called from vws_cadm.
It is also useful to have an idle window that will respond to UNIX commands.

An appropriate window setup may be made by using the name of the following 4-line suntools file as an argument to the "suntools" command used to set up the windows. You can use whatever name you like for the file, of course. A good choice might be "pde_windows".

```
cmdtool    -Wp   0   0 -Ws 650 647 -WP   75 836 -Wl "<< CONSOLE >>" -WL console -C
shelltool  -Wp  502 395 -Ws 650 505 -WP  1086 836 -Wi
shelltool  -Wp    0 670 -Ws 746 230 -WP  150 836 -Wl LISP
shelltool  -Wp  758   0 -Ws 394 839 -WP  300 836
```

The software for vws_cadm is included in a ready-to-use LISP environment that may be called as a UNIX command from an ordinary (shelltool) Sun window. The CPU of your Sun must have a floating-point calculation coprocessor in order to run the ready-to-use LISP environment. If it does not have this coprocessor, it will be necessary to build a new LISP environment.

The window you want to use for the vws_cadm command window must be running in a directory that contains the executable vws2_lisp file. This directory must have a subdirectory named "design" and must contain a copy of the file "font_pic". If you use the "pde_windows" file given above, then the window labelled "LISP" should be used as the vws_cadm command window.

Start the ready-to-use LISP environment in the command window by giving the command: vws2_lisp. Vws2_lisp will start up in a few seconds and will show the prompt "=>".

If you use the "pde_windows" file shown above, the window described on the last line of that file will be identified as "/dev/ttyp3". The Part Design Editor (PDE) is set to use "/dev/ttyp3" as the text window. If you wish to use a window with some other device number as the PDE text window (say "/dev/ttyp7"), it will be necessary to give the command: (setq pp_device "/dev/ttyp7") after starting vws2_lisp and before using PDE.

Vws_cadm should be started by responding to the "=>" prompt as follows: (vws_cadm)

4. USE OF FONTS IN THIS MANUAL

In order to make the operation of vws_cadm clear in this manual, different fonts will have different meanings in the remainder of the manual.

A. Discussion material will be set in this kind of type (ordinary Times font).

`B. Material printed in the command window by vws_cadm or other VWS2 system facilities will be set in this kind of type (ordinary Courier font).`

C. Material typed correctly to the command window by the user will be shown in this kind of type (boldface Times).

D. Material typed to the command window by the user that represents some kind of error (mipselling, for example) will be shown in this kind of type (boldface italic Times).

E. Headings: Section headings in the chapter on commands are printed in this kind of type (italic Times).

5. VWS_CADM OPTIONS

5.1. Introduction

As mentioned earlier, the following LISP command starts vws_cadm:

=> **(vws_cadm)**

When vws_cadm starts up, it prints a message and prompts the user as follows.

```
Reply as prompted by <> or b=break, h=help, q=quit, r=redo.
```

`<e p x i o>` We can use the help option to have vws_cadm explain these choices.

==> h

e = use design editor
p = make a process plan from an existing design
x = use existing process plan to write nc-code, verify, draw, or model
i = input a design, process plan, or workpiece
o = output a file or picture

<e p x i o>

==>

This gives the user nine options. The five options in angle brackets <e p x i o> are described in the commands section of this manual.

The other four options (break, help, quit, and redo) are available from the top level of vws_cadm, and also whenever being prompted by "==>" when dealing with four of the five main options (excluding the Design Editor). These four options are not available when giving input at any other prompt symbol.

Any entry other than these nine will cause vws_cadm to print an error message and prompt the user again.

5.2. Getting Help from vws_cadm

By entering "h" as shown in the preceding section, an explanation of the meaning of the options being given will be presented. After the help information is printed, the same prompt will be repeated.

5.3. Quitting

To get out of vws_cadm (or to abort whatever you are doing and pop up one level), enter "q". If you want to restart vws_cadm from LISP, enter (vws_cadm) after the LISP prompt.

Example 1:

`<e p x i o>` We call up the process planner.

==> p

Making a process plan from an existing design.
Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<design_id>

==> ***oops*** We have entered the wrong design id, so vws_cadm prompts us again.
`Can’t find design.  Please try again.`

<design_id>
We can’t remember the correct design id, so we decide to quit process planning.
==> **q**
`Returning to vws_cadm.`

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o> We decide to quit vws_cadm.

==> q
`Exiting vws_cadm.`
`t` Now we change our minds and start vws_cadm again.
=> **(vws_cadm)**

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>

==>

5.4. Redoing the Last Item

When using the "p" (process planning) and "x" (data execution) options, vws_cadm will ask a series of questions. If you make a mistake in answering a question, it may be corrected by entering "r" for redo. This will bring up the previous prompt message again. It is possible to back up several steps by entering "r" several times in a row. If you are at the top level of the "p" or "x" option, there is nothing to redo, so if you enter "r", vws_cadm just tells you it can't redo. The same "`Sorry. Cannot redo previous item.`" message appears any time an inappropriate "r" is entered after the "==>" prompt.

Example 1:

In this example, the user starts to use the process planner, then notices that the wrong design id was entered, so the user backs up (overshoots a little) and re-enters the design id.

`<e p x i o>`

==> **p**

`Making a process plan from an existing design.`
Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

`<design_id>`

==> ***lok2***

`<y or material>`

==> **y**

`<y = extras, n = no extras>`

==> **r**
`Redoing previous item.`

`<y or material>`

==> **r**
`Redoing previous item.`

`<design_id>`

==> ***r***
`Sorry.  Cannot redo previous item.`

`<design_id>`

==> **lok1**

`<y or material>`

The rest of the dialog (not shown) continues from here.

5.5. Breaking into LISP

To break into LISP, enter "b" after the "==>" prompt. To return to where you were, enter "?ret" to the LISP prompt. Do not break into LISP unless you know LISP. Be careful if you do break into LISP, since it is not hard to disrupt a LISP environment.

Example 1:

In this example, we are in the middle of a dialog with vws_cadm. We break out, execute a LISP command, and then return to the same point in the dialog.

```
<y or material>

==> b
Break: nil
c{1} (plus 1 2)
3
c{1} ?ret
<y or material>

==>
```

6. BUILDING VWS2_LISP

If vws2_lisp does not exist, or if you wish to rebuild it, it may be built as follows.

First, set up a directory with the following subdirectories and files taken from ~kramer/vws2:

Directories: cadm2, datab2, design2, draw2, exec2, geom2, misc2, plist2, proc2, vergen2, and verify2. The six subdirectories of datab2 (asn_c, asn_lisp, db_mgr, reports, rptgen, and world) must be present.

Files: font_pic, load_database.l, load_most.l, load_remob.l, load_subd.l, pde_windows

Set up a subdirectory named "design". This is a separate directory from "design2". The software for the Part Design Editor is in "design2". Designs created using the Editor will be stored in "design".

Edit the two files load_database.l and load_most.l by replacing "/usr2/kramer/vws2" with the path name of the directory you are working in.

Start LISP by giving the UNIX command: lisp.

When LISP is started, enter: (load_most).

This will build the environment. To save the environment, enter: (dumplisp vws2_lisp).
You can use some other name than "vws2_lisp" in that command if you wish. The name of the environment is automatically made into an executable UNIX command.

7. OTHER VWS2 CAPABILITIES

All major capabilities of the vws2_lisp environment may be accessed through vws_cadm. A couple minor capabilities which might have been made accessible have not. They may be accessed by the expert user by breaking into LISP. These include:

• the "pp_plist" LISP command for printing a nicely formatted copy of a property list. Most of the large data structures used by vws2_lisp are property lists.
• the "flash_step" LISP command which flashes the image of the change in the picture of a part caused by executing a step from a process plan

There are other capabilities of the VWS2 system, which are not in the vws2_lisp environment and cannot be accessed through vws_cadm. These include the control system and the state table editor, in particular. Most other VWS2 capabilities are accessible from the control system written by Dr. Jau-Shi Jun. That control system, which is also written in Franz LISP, has its own user-friendly interface.

When it is desired to have all capabilities of the VWS2 system available simultaneously, the usual procedure is to have the control system running in one window and vws2_lisp running in another.

The Franz LISP being used for the VWS2 system allows a LISP environment to grow to only 5 Mb. That is not enough room to load and operate all the software simultaneously. We believe that it would be feasible to load and operate all the software if the LISP environment were not limited to 5 Mb, but we have not tested this.

8. FORMAT OF COMMANDS SECTION

The commands section of this manual has five subsections, one for each of the five top-level vws_cadm commands. Two of the subsections are further divided. For each vws_cadm command the following information is given.

Purpose - What the command does.

Use - How to use the command and the circumstances in which to use it.

Error Handling - What the system does to prevent errors when the command is used and how the user can recover from errors in using the command.

Notes - Closely related commands, references, and notable peculiarities of the command.

Examples - One or more examples of the use of the command. In most cases, at least one example contains user errors. Errors are shown in a unique type font.

II. COMMANDS

1. e (Use the Part Design Editor)

Purpose:

To call up the Part Design Editor (PDE).

Use:

Use PDE when you want to create a new design or change an existing design. When the user quits from PDE, the top level prompt of vws_cadm reappears.

Error Handling:

No error handling.

Notes:

PDE has a separate users manual, which is Appendix B.

Example 1:

```
<e p x i o>
```

==> **e**

Entering design editor.

The screen clears.

pde >

A session with PDE follows, which is not shown here. The last command in the session is:

pde > **q**

Returning to vws_cadm.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>

==>

2. p (Make a Process Plan)

Purpose:

To use the Process Planning module to make a process plan from an existing design.

Use:

Vws_cadm conducts a dialog with the user to find out what the user wants to do. Then it runs the Process Planning module as directed. Notices are sent to the user when the module starts, and when it is done, but not while it is busy. Typical processing times for running the module are 10 seconds for a design with 5 features, none of which is a contour feature, and 5 minutes for a design with 25 features, including several elaborate contour features.

Error Handling:

Vws_cadm checks whether the given design has the right format for a design and prompts the user for another design, if not.

Vws_cadm checks whether the given material is one of: aluminum, brass, steel, or monel and prompts the user for another material, if not.

Vws_cadm checks whether the given plan name already has a property list in the LISP environment and prompts the user for another name, if so.

For both formats, if the user wants the plan saved in a file, after the user has proposed a file name, vws_cadm checks whether a file with the given name already exists and prompts the user for another name, if so.

Notes:

When the plan is written to a file, it is convenient to use a spare window running UNIX to look it over.

Example 1:

In this example, the user makes a process plan named "lok1_pl", stores it in LISP-readable form in a file named "lok1_pl2", and stores it in AMRF standard form in a file named "lok1_pl3". The plan is for making a part whose design is "lok1". The user makes a number of errors, asks for help once, and redoes one item.

`<e p x i o>`

`==>` **p**

`Making a process plan from an existing design.`
Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

`<design_id>`
`==>` ***look1***
`Can't find design.  Please try again.`

<design_id>
==> **lok1**

<y or material>
==> ***grass***
Material not known. Please try again.

<y or material>
==> **brass**

<y = extras, n = no extras>
==> **h**
If you want speeds, feed_rates and incremental cut depths to be put into the plan, enter y. If not, enter n.

<y = extras, n = no extras>
==> **y**

<plan_id>
==> **r**
Redoing previous item.

<y = extras, n = no extras>
==> **n**

<plan_id>
==> **lok1_plan**
Object with that name already exists. Please try again.

<plan_id>
==> **lok1_pl**

<plan_file_name or n>
==> **demo/lok1_plan**
That file already exists. Please try again.

<plan_file_name or n>
==> **lok1_pl2**

<flat_file_name or n>
==> **lok1_pl3**

I am starting to generate the process plan. Ten seconds pass.

The process plan has been generated.
Returning to vws_cadm.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.
<e p x i o>

3. x (Use the Data Execution Module)

Purpose:

To do any combination of the following 5 items (simultaneously):
A. verify a process plan
B. write an NC-code file to carry out a process plan
C. prepare an enhanced process plan
D. draw a picture to emulate the execution of process plan
E. get a description of the part resulting from the execution of a process plan

Use:

Vws_cadm engages the user in a fairly long dialog and then activates the Data Execution module as directed.

Error Handling:

The Data Execution module employs dozens of error checking routines. These are described in detail in [K&S1].

Vws_cadm checks that the process plan named by the user is a valid process plan. If the workpiece named by the user exists in the LISP environment, vws_cadm checks that it is a valid workpiece. It also checks whether any files the user wants to have written already exist. Existing files will not be overwritten.

Notes:

The Data Execution module is described in detail in [KR&W].

Example 1:

This is a simple example using only verification and NC-coding. A much longer example is given in the tutorial at the end of this manual.

```
<e p x i o>
```

==> **x**

```
The data execution module uses an existing plan.  It can verify the
plan, generate nc_code, draw the part, and make a model of the part.
```
Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

```
<plan_id>
```

==> **dog_plan**

```
<workpiece>
```

==> **dog_part**

```
<v = vise, p = pallet>
```
==> v

<f = fixture, t = top of part>

==> **f**

<s = on soft, v = on hard, o = off>

==> **s**

<nc_file_name or n>

==> **dognc**

<y = draw, n = don't>

==> **n**

<nhans_file_name or n>

==> **n**

<part_file_name or n>

==> **n**

Starting plan execution.
Starting design enhancement.
Design enhancement completed successfully.
Feature 1 side_contour is OK.
Feature 2 hole is OK.
Starting process plan enhancement, phase 1.
Process plan enhancement, phase 1, completed successfully.
Starting process plan enhancement, phase 2.
Process plan enhancement, phase 2, completed successfully.
Fixturing is OK.
Step 1 initialize_plan is OK.

EXECUTION OF THE PROCESS PLAN NAMED dog_plan
USING THE WORKPIECE NAMED dog_part

Automatic drawing is off.
Plan verification is on soft.
nc-code is being written to the file dognc.

Step 2 set0_corner is OK.
Step 3 mill_side_contour is OK.
Step 4 mill_pocket is OK.
Step 5 machine_countersink is OK.
Step 6 close_plan is OK.
Execution of the plan is completed.
Returning to vws_cadm.

4. i (Use Input Capabilities)

4.1. dl (Load a LISP-readable Design)

Purpose:

To load a LISP-readable design file into active memory.

Use:

Vws_cadm prompts the user for the name of the file. Vws_cadm looks for this file (with a ".pd" suffix added) in the "design" subdirectory. If the file is there, it is loaded.

In order to do anything with a design (make a process plan, execute a process plan, draw the design, etc.), it must be in active memory. This is one way to get a design into active memory.

Error Handling:

If the file cannot be found in the "design" subdirectory, the user is notified.

Notes:

This command is identical to the "l" command in the Part Design Editor. It adds the name of the design to the PART_DESIGN_LIST

Example 1:

```
<e p x i o>

==> i

This is the vws_cadm input facility.
Use it to load a design, process plan, or workpiece description.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

==> dl

Enter a part design file name ? tutorial
;; Loading file "./design/tutorial.pd"

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

==>
```

4.2. pl (Load a LISP-readable Process Plan)

Purpose:

To load a LISP-readable process plan file into active memory.

Use:

Vws_cadm prompts the user for the name of the file. The user must give the full UNIX path name. Vws_cadm looks for this file. If the file is found, it is loaded.

In order to use a process plan in the Data Execution module, it must be in active memory. This is one way to get a process plan into active memory. If a process plan has been created during a session with vws_cadm, it does not need to be loaded again.

Error Handling:

If the file is not found, the user is notified.

Notes:

The "pf" command described on the next page may be used to read in an AMRF standard format process plan.

Example 1:

```
<e p x i o>

==> i

This is the vws_cadm input facility.
Use it to load a design, process plan, or workpiece description.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

==> pl

Ready to load a lisp-readable process plan file.
Use a UNIX file path name.
Enter file name or q to quit. ? demo/lok1_plan
;; Loading file "demo/lok1_plan"

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

==> q
Returning to vws_cadm.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>
```

4.3. pf (Read in an AMRF Standard Format Process Plan)

Purpose:

To read a process plan file written in standard AMRF format into active memory.

Use:

Vws_cadm prompts the user for the name of the file. The user must give the full UNIX path name. Then vws_cadm prompts the user for the name to use for the plan in active memory. Vws_cadm looks for the specified file. If the file is found, it is read in. The reader parses it into the LISP structure required by the VWS2 system.

In order to use a process plan in the Data Execution module, it must be in active memory. This is one way to get a process plan into active memory. If a process plan has been created during a session with vws_cadm, it does not need to be read in again.

Error Handling:

If the file is not found, the user is notified.

Notes:

The "pl" command described on the preceding page may be used to load a LISP-readable process plan.

Example 1:

`<e p x i o>`

==> **i**

`This is the vws_cadm input facility.`
Use it to load a design, process plan, or workpiece description.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

`<dl pl pf wl>`

==> pf

`Ready to read an AMRF standard process plan file.`
Use a UNIX file path name.
`Enter file name or q to quit. ?` **tutorial_plan_std**
`Enter plan name. ?` **tutorial_plan**

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

`<dl pl pf wl>`

4.4. wl (Load a LISP-readable Workpiece Description)

Purpose:

To load a LISP-readable workpiece description from a file into active memory.

Use:

Vws_cadm prompts the user for the name of the file. The user must give the full UNIX path name. Vws_cadm looks for this file. If the file is found, it is loaded.

In order to draw a workpiece or use an existing workpiece in the Data Execution module, it must be in active memory. This is one way to get a workpiece description into active memory.

Error Handling:

If the file is not found, the user is notified.

Notes:

No notes.

Example 1:

`<e p x i o>`

==> **i**

This is the vws_cadm input facility.
Use it to load a design, process plan, or workpiece description.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

==> **wl**

Ready to load a lisp-readable workpiece description file
Use a UNIX file path name.
`Enter file name or q to quit. ?` **demo/darpa_part_in**
`;; Loading file "demo/darpa_part_in"`

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

5. o (Use Output Capabilities)

5.1. pl (Print a LISP-readable Process Plan File)

Purpose:

To save a LISP-readable process plan in a file.

Use:

If you make a process plan with the Process Planning module, but do not write it to a file at the time, and later you decide to save it in LISP-readable form in a file, this is the command to use. If a process plan is available in AMRF standard format, and you want it in a file in LISP-readable format, it may be read in from the standard format and then printed with this command.

The user is prompted for the plan name and for the name of the file into which the plan should be written.

Error Handling:

If no data structure with the given name is in the environment, or if a data structure with the given name which is not a process plan is in the environment, an error message is printed.

If a file already exists with given file name, the user is asked if it is OK to overwrite it.

Notes:

Use the "pf" command described on the next page to save a process plan in AMRF standard format.

Example 1:

```
<e p x i o>

==> o

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> pl

Ready to print a lisp-readable process plan file.
The process plan must exist in the Lisp environment.
Enter process plan name or q to quit. ? tutorial_plan
Enter UNIX file name or q to quit. ? tut_plan
```

The file is written, and the user is prompted for more input.

5.2. pf (Print an AMRF Standard Format Process Plan File)

Purpose:

To save a process plan in AMRF standard format in a file.

Use:

If you make a process plan with the Process Planning module, but do not write it to a file at the time, and later you decide to save it in AMRF standard form in a file, this is the command to use. If a process plan is available in a LISP-readable file, and you want it in a file in AMRF standard format, load the LISP-readable file, and then print the standard format file with this command.

The user is prompted for the plan name and for the name of the file into which the plan should be written.

Error Handling:

If no data structure with the given name is in the environment, or if a data structure with the given name which is not a process plan is in the environment, an error message is printed.

If a file already exists with given file name, the user is asked if it is OK to overwrite it.

Notes:

Use the "pl" command described on the preceding page to save a process plan in LISP-readable format.

Example 1:

`<e p x i o>`

`==>` **o**

`This is the vws_cadm output facility.`
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

`Reply as prompted by <> or b=break, h=help, q=quit, r=redo.`

`<pl pf dl dh gd gw gt ge>`

`==>` **pf**

`Ready to print an AMRF standard process plan file.`
The process plan must exist in the Lisp environment.
`Enter process plan name or q to quit. ?` **tutorial_plan**
`Enter UNIX file name or q to quit. ?` **tut_plan_std**
`That file already exists.  Do you want to overwrite it?`
Enter yes or no. ? **yes**

The file is written, and the user is prompted for more input.

5.3. dl (Print a LISP-readable Design File)

Purpose:

To save a design in LISP-readable form.

Use:

The user is prompted for the name of the design. A file whose name is the name of the design with the suffix ".pd" is written in the "design" subdirectory. This command might be used if you edited a design with the Part Design Editor, but did not save it at the time.

Error Handling:

If no data structure with the given name is in the environment, or if a data structure with the given name which is not a design is in the environment, an error message is printed.

If a file already exists with given file name, the user is asked if it is OK to overwrite it.

Notes:

There is rarely a need to use this command.

The "dh" command described on the next page may be used to write a design to a file in human-readable form.

Example 1:

`<e p x i o>`

`==>` **o**

`This is the vws_cadm output facility.`
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

`Reply as prompted by <> or b=break, h=help, q=quit, r=redo.`

`<pl pf dl dh gd gw gt ge>`

`==>` **dl**

Ready to print a lisp-readable design file.
The design must exist in the Lisp environment.
`Enter design name or q to quit. ?` **dog**
`The file design/dog.pd already exists.`
`Do you want to overwrite it?`
Enter yes or no. ? **yes** The file is written.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

5.4. dh (Print a Human-readable Design File)

Purpose:

To save a design in human-readable form.

Use:

The user is prompted for the name of the design. A file whose name is the name of the design with the suffix ".prt" is written in the "design" subdirectory. This command might be used if you edited a design with the Part Design Editor, but did not save it at the time.

Error Handling:

If no data structure with the given name is in the environment, or if a data structure with the given name which is not a design is in the environment, an error message is printed.

If a file already exists with given file name, the user is asked if it is OK to overwrite it.

Notes:

There is rarely a need to use this command.

The "dl" command described on the preceding page may be used to write a design to a file in LISP-readable form.

Example 1:

`<e p x i o>`

`==>` **o**

`This is the vws_cadm output facility.`
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

`Reply as prompted by <> or b=break, h=help, q=quit, r=redo.`

`<pl pf dl dh gd gw gt ge>`

`==>` **dh**

Ready to print a human-readable design file.
The design must exist in the Lisp environment.
`Enter design name or q to quit. ?` **dog**
`The file design/dog.prt already exists.`
`Do you want to overwrite it?`
Enter yes or no. ? **yes** The file is written.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

5.5. gd (Draw a Design)

Purpose:

To draw the picture of a design.

Use:

If you want to see what a design looks like, but do not want to edit the design, this command may be used. The description of the design must be present in the environment. The user is prompted for the name of the design and the level of verification which should be used.

Error Handling:

If no data structure with the given name is in the environment, or if a data structure with the given name which is not a design is in the environment, an error message is printed.

Notes:

A design description may have been created or loaded by the Part Design Editor, or it may be loaded with the "dl" command of the vws_cadm input facility.

It is usually more useful to use the Part Design Editor to draw designs.

Example 1:

```
<e p x i o>

==> o

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> gd

Ready to draw a design.
The design must exist in the Lisp environment.
Enter design name or q to quit. ? dog
Enter verification level: off, soft or hard. ? soft
Feature 1 side_contour is OK.
Feature 2 hole is OK.

The picture (not shown here) is drawn.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>
```

5.6. gw (Draw a Workpiece)

Purpose:

To draw a picture of a workpiece

Use:

If you want to see what a workpiece looks like, use this command. The description of the workpiece must be present in the environment. The user is prompted for the name of the workpiece and the level of verification which should be used.

Error Handling:

If no data structure with the given name is in the environment, or if a data structure with the given name which is not a workpiece is in the environment, an error message is printed.

Notes:

A workpiece description may have been created by the Data Execution module, or it may be loaded with the "wl" command of the vws_cadm input facility.

Example 1:

`<e p x i o>`

==> **o**

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **gw**

Ready to draw a workpiece.
The workpiece must exist in the Lisp environment.
Enter workpiece name or q to quit. ? ***yoke_part_in***
`That is not a proper workpiece.`

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

Example 2:

<e p x i o>

==> **o**

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

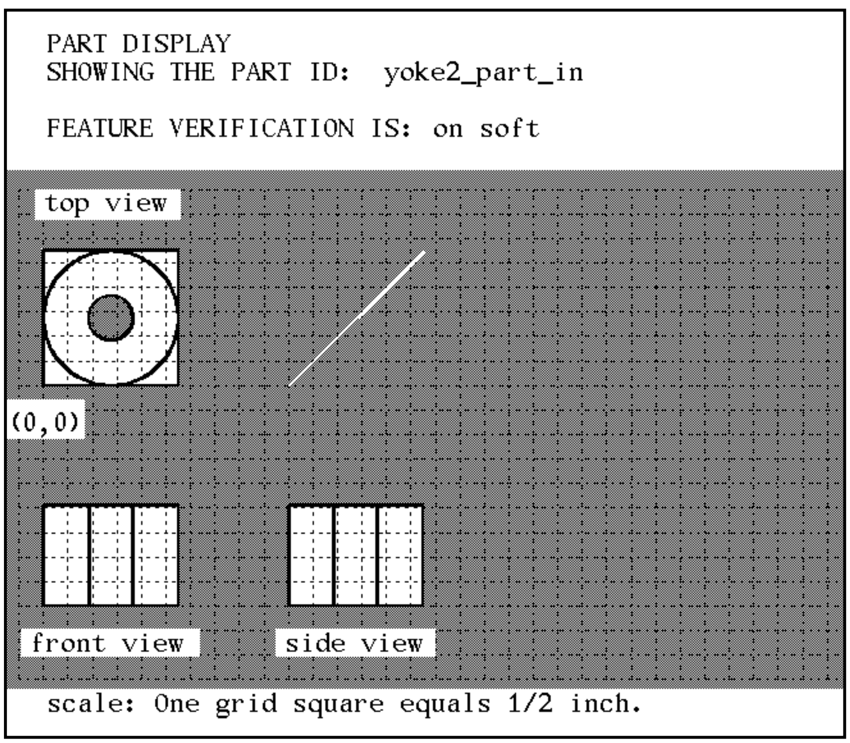
==> **gw**

Ready to draw a workpiece.
The workpiece must exist in the Lisp environment.
Enter workpiece name or q to quit. ? **yoke2_part_in**
Enter verification level: off, soft or hard. ? ***yes***
Enter verification level: off, soft or hard. ? **soft**
Feature 1 side_contour is OK.
Feature 2 hole is OK.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

An example of a picture drawn by this facility is shown at the right.



5.7. gt (Draw Tool Path on Existing Picture)

Purpose:

To draw the tool path described by an NC-program

Use:

This command is available only after the Data Execution module has been run with drawing and NC-coding options on. It is used to check out the NC-code. When the tool is moving at cutting speed a thin dotted line is drawn. When the tool is moving at traverse speed (it better not be cutting), a heavy dotted line is drawn.

The user may choose to have the picture drawn slowly or to break between cuts.

Error Handling:

If it is not possible to draw the tool path, the message "There is no tool path to draw." will be sent.

Notes:

A fuller description of tool path drawing is given in [K&S1], chapter XII.

Example 1:

`<e p x i o>`

==> **o**

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **gt**

There is no tool path to draw.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

Example 2:

<e p x i o>

==> **o**

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **gt**

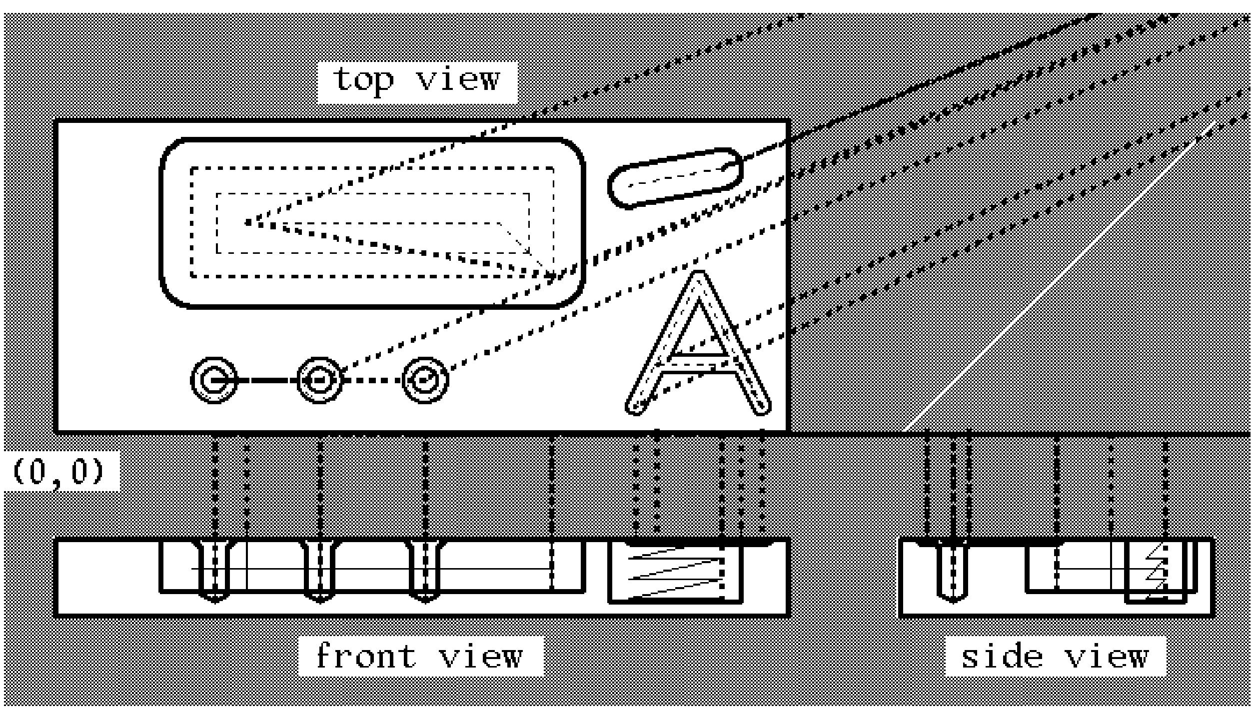
Break between cuts (y/n) ? **n**

Draw slowly (y/n) ? **n**

A tool path is drawn. An example is shown below.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>



5.8. ge (Erase Existing Picture)

Purpose:

To erase any existing picture drawn by the VWS2 system.

Use:

Use this command to get rid of the picture made by the Data Execution module, the design drawer, or the workpiece drawer. The Part Design Editor normally erases its own pictures, but if one of those remains on the screen after the Editor exits, this command will get rid of it.

Error Handling:

No error handling. If there is no picture, the command does nothing.

Notes:

This command will not affect pictures which are not drawn by the system.

Example 1:

`<e p x i o>`

==> **o**

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **ge**

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

III. VWS_CADM TUTORIAL

This is a tutorial for you to use to train yourself in using vws_cadm. The convention regarding the use of fonts followed in this tutorial was described in section 4 of the introduction to this manual. To use the tutorial, log in to your Sun and be sure your environment is set up as described in section 3 of the introduction. Do not start a window system yet.

This tutorial will have you use four of the five vws_cadm facilities. The use of the Part Design Editor (PDE) is not described here, since there is a separate users manual for PDE. You will:
• load an existing design
• make a process plan for milling a part of that design and save the plan in two formats
• write NC-code for milling the part
• make a graphical emulation of the milling process
• verify the process plan
• save the enhanced process plan
• save a description of the finished part
• draw the tool path used to cut the part

The design to be loaded is named "tutorial". Be sure the file "tutorial.pd" is in your "design" subdirectory before starting. If it isn't, do the tutorial in the PDE Users Manual first.

First, enter the following on the terminal (which should be using the full screen at this point).

suntools pde_windows

This should format your screen with a set of windows. Put the mouse cursor in the window labelled LISP and enter the following:

vws2_lisp

In a few seconds, LISP should start up and show you an arrow prompt. From here to the end of the tutorial, enter the text shown in **boldface Times font** in your command screen. All the following material shown in `courier font` should be printed on your screen by the system. Explanatory material is in this kind of font.

Wherever this tutorial says you should enter **h** for help, you may skip to the next entry if you like.

If you make an error, use the "redo" command or read about the command in which you made the error in the commands section of this manual to try to recover. Good luck.

=> **(vws_cadm)**

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>

First we load the design by using the input facility. Then we get out of the input facility.

==> **i**

This is the vws_cadm input facility.
Use it to load a design, process plan, or workpiece description.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

==> **dl**

Enter a part design file name ? **tutorial**
`;; Loading file "./design/tutorial.pd"`

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<dl pl pf wl>

==> **q**
`Returning to vws_cadm.`

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>

Now we make a process plan and print it to a file in LISP-readable format.

==> **p**

Making a process plan from an existing design.
Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<design_id>

==> **tutorial**

<y or material>

==> **h**
If you want to plan for the material specified in the design, enter y. Otherwise, enter aluminum, brass, steel, or monel.

<y or material>

==> **y**

<y = extras, n = no extras>

==> **h**
If you want speeds, feed_rates and incremental cut depths to be put into the plan, enter y. If not, enter n.

<y = extras, n = no extras>

==> **n**

<plan_id>

==> **h**
A new plan will be created and kept in the LISP environment. Enter the plan id of the plan to be written.

<plan_id>

==> **tutorial_plan**

<plan_file_name or n>

==> **h**
If you want to save the plan in a lisp_readable file, enter the file name. If not, enter n.

<plan_file_name or n>

==> **tutorial_plan**

<flat_file_name or n>

==> **h**
If you want to save the plan in standard AMRF format, enter the file name. If not, enter n.

<flat_file_name or n>

==> **n**

```
I am starting to generate the process plan.
```

A minute passes with nothing happening on the screen.

The process plan has been generated.
Returning to vws_cadm.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>

Next, we execute the process plan. We decide to use all five options of Data Execution module: write NC-code, draw a picture, save the enhanced plan, verify the plan, and save a description of the finished part.

==> **x**

The data execution module uses an existing plan. It can verify the plan, generate nc_code, draw the part, and make a model of the part.
Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<plan_id>

==> **tutorial_plan**

<workpiece>

==> **h**

```
Please enter a workpiece id.  It may be a defined workpiece, in
which case workpiece material must match plan material.  If
workpiece is not defined, a definition will be created using the
block size from the design and the material from the plan.
```

<workpiece>

==> **tutorial_part**

<v = vise, p = pallet>

==> **h**

```
If the workpiece is to be fixtured in the vise, enter v.
If the workpiece is to be fixtured on a pallet, enter p.
```

<v = vise, p = pallet>

==> **v**

```
<f = fixture, t = top of part>
```

==> **h**
```
If the bottom of the vise or pallet is to be used as the z_reference
plane, enter f.  If the top of the workpiece is to be used as the
z_reference plane, enter t.
```

<f = fixture, t = top of part>

==> **f**

<s = on soft, v = on hard, o = off>

==> **h**
```
If you want the plan verified, enter s or v.  If not, enter o. If
s is chosen and an error is detected, you will have the option to
continue or abort.  If v is chosen, an error always aborts. The
design will also be verified if s or v is chosen. It is usually
wise to verify.
```

<s = on soft, v = on hard, o = off>

==> **s**

<nc_file_name or n>

==> **h**
```
If you want to generate nc-code to make the part, enter the name
of the file in which the code should appear.  If not, enter n.
```

<nc_file_name or n>

==> **tutonc**

<y = draw, n = don't>

==> **h**
```
If you want a picture of the part, enter y. If not enter n.
```
The picture is a simulation of machining and will be drawn
in the order the plan is carried out.

<y = draw, n = don't>

==> **y**

```
<nhans_file_name or n>
```

==> **h**
If the process plan has already been enhanced as described below, enter t. Otherwise, an enhanced version of the process plan will be prepared. This version contains changer slot numbers, speeds, feed-rates, and pass-depths for each step. Also, face-milling may be added if the workpiece is too high, various zero-setting steps will be added, and steps may be deleted if the workpiece already has features in it. If you want a copy of this enhanced process plan, enter the name of the file in which the enhanced plan should appear. If not, enter n.

<nhans_file_name or n>

==> **tutorial_plan_nhans**

<part_file_name or n>

==> **h**

```
If you want to save a description of the finished part in the LISP
environment and in a file, enter the file name. If not, enter n.
```

<part_file_name or n>

==> **tutorial_part**

```
Starting plan execution.
Starting design enhancement.
```

Design enhancement completed successfully.
Feature 1 hole is OK.
Feature 2 hole is OK.

Sixteen more messages (omitted here) are printed, saying that features 3 through 18 are OK.

Starting process plan enhancement, phase 1.
Process plan enhancement, phase 1, completed successfully.
Starting process plan enhancement, phase 2.
Process plan enhancement, phase 2, completed successfully.
Fixturing is OK.
Step 1 initialize_plan is OK.

The graphics window is started up. It is not shown in this tutorial. After each step is verified as shown below, the picture changes to show the feature or subfeature which is added by the step.

Two of the steps fail verification. We tell the system to go ahead anyway. The error messages are real. The NC-program we are writing should not be run because it will mill into the vise in three places.

Step 2 set0_corner is OK.
Step 3 mill_contour_pocket is OK.
Step 4 mill_contour_pocket is OK.

Tool hits side obstacle in mill_st_groove_test.
Tool hits side obstacle in mill_st_groove_test.
Step 5 mill_straight_groove fails operation verifier.
To attempt to continue at risk of further error, enter "go".
Otherwise hit any key and a return.
==> **go**

Step 6 mill_groove is OK.
Step 7 mill_groove is OK.

A slew of messages (omitted here) are printed saying that steps 8 through 36 are OK.

Drill hits bottom obstacle in drill_hole_test.
Step 37 drill_hole fails operation verifier.
To attempt to continue at risk of further error, enter "go".
Otherwise hit any key and a return.
==> **go**

Step 38 close_plan is OK.
Execution of the plan is completed.
To wipe out the picture, break and give the command (close_part_display).
Returning to vws_cadm.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>

Finally, we use the vws_cadm output facility to save the process plan in another form, to draw the tool path, and to erase the picture.

==> **o**

This is the vws_cadm output facility.
Use it to print a design or a process plan, or to
draw or erase a picture of a design or a workpiece.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **h**
`pl = print a lisp-readable process plan file`
pf = print an AMRF standard format process plan file
dl = print a lisp-readable design file in the design directory
dh = print a human-readable design file in the design directory
gd = draw a design
gw = draw a workpiece
gt = draw tool path on existing picture
ge = erase existing picture

<pl pf dl dh gd gw gt ge>

==> **pf**

Ready to print an AMRF standard process plan file.
The process plan must exist in the Lisp environment.
`Enter process plan name or q to quit. ?` **tutorial_plan**
`Enter UNIX file name or q to quit. ?` **tutorial_plan_std**

The plan file is written. Next we draw the tool path.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **gt**

Break between cuts (y/n) ? **n**

Draw slowly (y/n) ? **n**

The tool path is drawn on the existing picture. This takes a while.
After studying the tool path a little, we erase the picture.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **ge**

The picture is erased. Next we get out of the output facility and out of vws_cadm.

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<pl pf dl dh gd gw gt ge>

==> **q**
```
Returning to vws_cadm.
```

Reply as prompted by <> or b=break, h=help, q=quit, r=redo.

<e p x i o>

==> **q**
```
Exiting vws_cadm.
t
```

To end, we exit from LISP.

=> **(exit)**

If you wish to look at any of the files written during this tutorial, use a UNIX command like:

more tutonc

The files which were written are as follows:

tutonc	NC-code
tutorial_plan	LISP-readable process plan
tutorial_plan_std	AMRF standard format process plan
tutorial_plan_nhans	LISP-readable enhanced process plan
tutorial_part	LISP-readable description of the finished part

REFERENCES

[K&S1]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention and Detection in Data Preparation for the Vertical Workstation Milling Machine in the Automated Manufacturing Facility at the National Bureau of Standards"; NBSIR 87-3677; National Bureau of Standards; 1987; 61 pages.

[KR&W]
Kramer, Thomas R.; and Weaver, Rebecca E.; "The Data Execution Module of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3704; National Bureau of Standards; 1988; 58 pages.

[SUN1]
"Getting Started with UNIX: Beginner's Guide"; Sun Microsystems; Part No: 800-1284-03; 1986; 143 pages.

[SUN2]
"Windows and Window Based Tools: Beginner's Guide"; Sun Microsystems; Part No: 800-1287-03; 1986; 186 pages.

APPENDIX B

DESIGN EDITOR USERS MANUAL

Dr. Thomas R. Kramer
Guest Worker, National Bureau of Standards, &
Research Associate, Catholic University

March 10, 1988

CONTENTS

Page

I. INTRODUCTION B-1
1. CONTENTS B-1
2. AUDIENCE B-1
3. FORMAT OF COMMANDS SECTION B-1
4. STARTING THE PART DESIGN EDITOR B-2
5. USE OF FONTS IN THIS MANUAL B-3
6. NOTES B-4
6.1. Authorship B-4
6.2. Input to PDE B-4
6.3. Undoing B-4
6.4. Verification B-4
6.5. Design for Machining B-5

II. COMMANDS B-6
array B-6
b (break) B-9
block B-10
c (change) B-11
clp B-15
cls B-16
d (delete) B-17
dpd B-19
draw B-20
e (edit) B-21
elem B-22
feat B-23
flash B-24
goff B-25
gon B-26
group B-27
i (insert) B-29
l (load) B-31
lff B-32
loc B-33
ls B-35
m (menu) B-36
n (name) B-37
new B-38
p (print) B-40

pick....B-41
q (quit)....B-42
rdraw....B-43
rseq....B-44
save....B-45
stf....B-46
verif....B-47
vset....B-48

III. PART DESIGN EDITOR TUTORIAL....B-49

REFERENCES....B-60

I. <u>INTRODUCTION</u>

1. CONTENTS

This is a users manual for the Part Design Editor (PDE) module of the VWS2 software for the AMRF Vertical Workstation (VWS). It has two further sections: a guide to PDE commands, and a tutorial demonstration of PDE. You should try the tutorial after you read the introduction.

2. AUDIENCE

This manual is intended to be used by anyone learning to use PDE or using PDE. A person who is learning PDE should first read Chapter III "Part Design Editor" of the paper [KR&J] which describes the design protocol and the Part Design Editor. The first five pages of that chapter give a general description of PDE which is not repeated here. The remainder of that paper includes detailed descriptions of the features used in the VWS2 system, which should also be kept nearby for reference.

Before trying to learn PDE, a person should be familiar with the basics of UNIX and the basics of the window system "suntools" which is part of the standard software for a Sun computer. Both are easy to learn. A good manual to use to learn enough about UNIX is <u>Getting Started with UNIX: Beginner's Guide</u> [SUN1]. A good manual to use to learn enough about the window system is <u>Windows and Window Based Tools: Beginners Guide</u> [SUN2].

To use PDE, It is useful but not necessary to know the computer language LISP, since the entire VWS2 system is written in LISP.

3. FORMAT OF COMMANDS SECTION

The command section of this manual is arranged alphabetically by command name. For each command the following sections are given.

Purpose - What the command does.

Use - How to use the command and the circumstances in which to use it.

Error Handling - What the system does to prevent errors when the command is used and how the user can recover from errors in using the command.

Notes - Closely related commands, references, and notable peculiarities of the command.

Examples - One or more examples of the use of the command. In most cases, at least one example contains user errors. Errors are shown in a unique type font. For many commands, when it is finished executing the command, PDE prints the descriptions of four features in the text window. This printing is omitted from most examples.

4. STARTING THE PART DESIGN EDITOR

The Part Design Editor (PDE) should be run on a Sun with at least 4Mb of on-board memory. It will run slowly unless there are at least 6Mb of on-board memory.

PDE runs in a suntools window environment. The screen should be set up with at least three windows:
A. A Console window to intercept messages from outside.
B. A PDE command window in the lower left of the screen.
C. A PDE text window covering the right-hand third of the screen from the top to near the bottom.
It is also useful to have an idle window that will respond to UNIX commands.

An appropriate window setup may be made by using the name of the following 4-line suntools file as an argument to the "suntools" command used to set up the windows. You can use whatever name you like for the file, of course. A good choice might be "pde_windows".

```
cmdtool    -Wp   0   0 -Ws 650 647 -WP   75 836 -Wl "<< CONSOLE >>" -WL console -C
shelltool  -Wp  502 395 -Ws 650 505 -WP  1086 836 -Wi
shelltool  -Wp    0 670 -Ws 746 230 -WP  150 836 -Wl LISP
shelltool  -Wp  758   0 -Ws 394 839 -WP  300 836
```

The software for PDE is included in a ready-to-use LISP environment that may be called as a UNIX command from an ordinary (shelltool) Sun window. The CPU of your Sun must have a floating-point calculation coprocessor in order to run the ready-to-use LISP environment. If it does not have this coprocessor, it will be necessary to build a new LISP environment.

The window you want to use for the PDE command window must be running in a directory that contains the executable vws2_lisp file. This directory must have a subdirectory named "design" and must contain a copy of the file "font_pic". If you use the "pde_windows" file given above, then the window labelled "LISP" should be used as the PDE command window.

Start the ready-to-use LISP environment in the command window by giving the command: vws_lisp. Vws2_lisp will start up in a few seconds and will show the prompt "=>".

If you use the "pde_windows" file shown above, the window described on the last line of that file will be identified as "/dev/ttyp3". PDE is set to use "/dev/ttyp3" as the text window. If you wish to use a window with some other device number as the PDE text window (say "/dev/ttyp7"), it will be necessary to give the command: (setq pp_device "/dev/ttyp7") after starting vws2_lisp and before starting PDE.

PDE may be started by responding to the "=>" prompt in one of two ways:
A. Enter: (pde)
B. Enter: (vws_cadm) This will bring up another prompt "==>". Enter: e

5. USE OF FONTS IN THIS MANUAL

In order to make the operation of PDE clear in this manual, different fonts will have different meanings in the remainder of the manual.

A. Discussion material will be set in this kind of type (ordinary Times font).

B. Material printed on the screen by PDE will be set in this kind of type (ordinary Courier font).

```
If material in Courier font
appears in a box like this,
that means it appears in the
PDE text window. The box
will appear in the order in
which action is occurring
on the screen. When you are
using PDE, the material in
a box in this manual
actually appears on the
right side of the screen, or
wherever you put the PDE
text window.
```

If material in Courier font is not in a box, that means it is printed to the command window by PDE.

C. Material typed correctly to the command window by the user will be shown in this kind of type (boldface Times).

D. Material typed to the command window by the user that represents some kind of error (mipselling, for example) will be shown in this kind of type (boldface italic Times).

E. Headings: Section headings in the chapter on commands are printed in this kind of type (italic Times).

Additional fonts appear on the PDE graphics screen from time to time. They are shown in pictures in this manual just as they appear on the screen.

The font actually used by PDE will be the same in the command window as in the text window, unless you customize one of them in your sunwindows setup file.

6. NOTES

6.1. Authorship

The PDE software itself was written jointly by Dr. Jau-Shi Jun and the author.

6.2. Input to PDE

All commands to PDE should be given in the PDE command window. The PDE text window is actually running a shell that responds to UNIX commands, so if you give PDE commands in the text window, you will get a confused reaction from UNIX. If you give PDE commands to the PDE graphics window, they will be ignored.

Some PDE commands will direct you to point at an object or location on the graphics window and push a button on the mouse. When you move the mouse cursor into the graphics window, the shape of the cursor will change to a hand with a pointing index finger. The tip of the index finger is at the point which will be picked by the mouse.

6.3. Undoing

There is no general "undo" command in PDE which will return you to the situation you were in with PDE before something went awry. You will often wish there were such a command. Sorry.

Some individual PDE commands have the ability to undo things. These abilities are discussed as part of the documentation of the command, in the "Error Handling" section.

6.4. Verification

The verification system is described in detail in [K&S1] and [K&S2]. Normally, PDE is run with verification "on_soft". In this verification mode, if an error occurs, the user is often given a choice of trying to proceed (by entering "go") or not (by entering anything else). In most cases it is best to enter "go" when given a choice. In some cases this will enable the user to see the error, so that it is easier to correct. In other cases it may save time, since correcting an error usually involves changing the value of one parameter, but redoing a dialog may entail entering ten or more parameter values.

If the verification system catches an error, the rule(s) which has been violated will be printed on the command window. The rules are printed in full in [KR&J].

PDE itself makes many checks of parameters when it is in use. It will usually prompt the user to enter something else if it finds an error.

6.5. Design for Machining

How to make a part of a given design, once the design is created, is described in the users manual for vws_cadm, which is Appendix A. There are two data preparation stages. First a process plan must be made, and then NC-code must be generated. Once the NC-code is in hand, it may be downloaded to the milling machine controller. A workpiece is then fixtured on the milling machine, and the NC-code is run, making the part.

Process planning uses a catalog of hypothetical tools. Process planning will fail if a tool cannot be found in the tool catalog to make some feature or subfeature in the design.

NC-coding uses a database of tools actually present on the milling machine. NC-coding will fail if a tool which is called for in the process plan is not on the milling machine. NC-coding may also fail if the fixturing interferes with a cutting path, or if a feature is too deep to be cut by the tool selected for it.

PDE does not constrain the range of possible designs to only those for which NC-code can be written. Here are some general rules which will help with the designing process if you want to design something and make it without changing the machine setup.

A. Never leave a feature which fails verification in a design.

B. Do not make conical_bottomed holes with arbitrary diameters. Such holes must be made by drilling in the VWS2 system, so know what size drills are actually available, and match the hole diameter to the drill diameter.

C. Do not make flat or round-bottomed features whose depth is more than three times the narrowest horizontal dimension.

D. Do not make conical-bottomed holes or through-holes more than six times as deep as the diameter of the hole.

E. When choosing the line width for text, PDE requires the user to select a width which can be made by some ball-nosed-end-mill in the tool catalog. Know which of those that are in the catalog are also on the milling machine.

F. Choose widths for all three types of grooves which are the widths of end mills on the milling machine.

G. Avoid round-bottomed grooves which are less than half as deep as they are wide. The chances of guessing the correct width for a given depth which matches a tool on the milling machine are very near zero.

H. Don't make threaded holes unless you know what properly matched drill-tap pairs are available on the milling machine.

I. Have an idea where the fixturing is going to be in contact with the part you are designing, and avoid putting features in those places.

II. COMMANDS

array

Purpose:

To make a feature pattern. This allows the user to generate an array of features identical to an existing feature except for location. Any feature may be duplicated in an array, but it makes little sense to have an array of side_contours or chamfer_outs. The array may be either rectangular or circular.

Use:

This causes PDE to enter into a substantial dialog with the user. The first choice is the feature to be duplicated, and the second choice is the type of array: rectangular or circular. Dialogs are shown in the examples.

Error Handling:

When the dialog with the user is finished, just before the new features are created, PDE asks the user if the dialog was correct. If the user replies "no", PDE repeats the dialog. When PDE asks, look carefully at your answers, which should still be showing in the command window. If there is an error, answer "no".

If one of the features being created in the array fails verification, or if some other type of error occurs, PDE returns the design to the way it was before "array" was called. This includes regenerating the entire picture of the design, which may take a long time. In order to avoid the wait, it is usually worth trying to proceed in spite of an error and going back and changing the offending feature after the array is complete. If the error is so bad that PDE cannot deal with it, the design will be restored as just described.

If the feature being duplicated does not exist, this cannot be corrected by redoing the dialog, so be sure the feature is identified correctly. PDE will recover from this error automatically, but it may take some time.

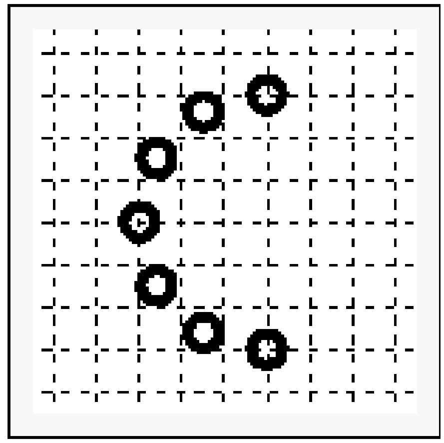
Notes:

A. When "array" is used, the new features are inserted immediately after the feature being duplicated. Thus, if the feature being duplicated is feature 5, and there are four features in the array, the new features are numbers 6, 7, and 8 (the existing feature is the first member of the array). After the insertions, the design is resequenced automatically, so that features which followed the feature being duplicated will have higher numbers than before.

B. After an array is created, it has no continuing existence as an array. It is just a bunch of individual features.

C. The "group" command also may be used to create a bunch of features automatically.

D. It may be convenient to use the "pick" command to identify the feature to be duplicated.

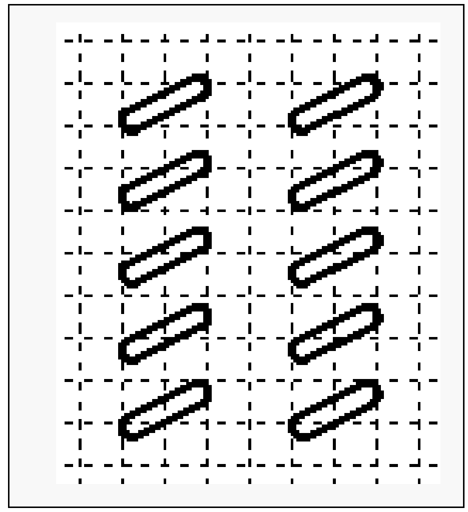


Example 1:

A semicircular array of holes may be created as follows. A picture of the top view of the array is shown at the right. Feature 1 is the hole at the top. It must exist before the array command is called.

```
pde > array
Enter feature number to be repeated ? 1

An array of features may be rectangular or in a circular arc.
Choose a type (1=rectangular  2=circular). ? 2

You will need to specify:
   1. number of features in the array,
   2. center_x and center_y of the circle,
   3. angle of the arc in degrees (360 for full circle).
   4. clockwise or counterclockwise.

Enter no. of times to be repeated in a circle ? 7
Enter the center_x ? 1.5
Enter the center_y ? 1.75
Enter the total angular (degree/(d)efault=360) ? 180
Enter (1) clockwise (2) counterclockwise ? 2

Is this correct (y/n) ? y
Feature i2 hole is OK.
Feature i3 hole is OK.
Feature i4 hole is OK.
Feature i5 hole is OK.
Feature i6 hole is OK.
Feature i7 hole is OK.

Done
```

Example 2:



A rectangular array of straight_grooves may be created as follows. A picture of the top view of the array is shown at the right. Feature 2 is at the top left of the array. It must exist before the array command is called.

```
pde > array
Enter feature number to be repeated ? 2

An array of features may be rectangular
     or in a circular arc.
Choose a type
    (1=rectangular  2=circular). ? 1

Array size is given either by the total x and y distances,
or by the incremental x and y distances between features.
Choose a method (1=incremental  2=total). ? 1

You will also need to specify the number of times the
feature is to be repeated in the x and y directions.

Enter no. of times to be repeated in x-direction ? 2
Enter no. of times to be repeated in y-direction ? five
Enter no. of times to be repeated in y-direction ? 4
Enter x-displacement ? 1
Enter y-displacement ? -0.5
Is this correct (y/n) ? n

Enter no. of times to be repeated in x-direction ? 2
Enter no. of times to be repeated in y-direction ? 5
Enter x-displacement ? 1
Enter y-displacement ? -0.45
Is this correct (y/n) ? y

Feature i3 straight_groove is OK.
Feature i4 straight_groove is OK.
Feature i5 straight_groove is OK.
Feature i6 straight_groove is OK.
Feature i7 straight_groove is OK.
Feature i8 straight_groove is OK.
Feature i9 straight_groove is OK.
Feature i10 straight_groove is OK.
Feature i11 straight_groove is OK.

Done
```

<u>b</u>

Purpose:

To put the user into the LISP command interpreter in the environment in which PDE is running. The command name "b" is an abbreviation of the LISP function "break".

Use:

Do not use this command if you do not know LISP. After this command has been given, any valid LISP commands may be given. One reason for getting into LISP is to do numerical calculations needed for specifying the parameters of features (although most people would find using a calculator program in a spare window easier). It is sometimes convenient to get into LISP to do editing operations (such as copying features from one design to another) not currently in PDE, but only users very familiar with the PDE software should try this.

To return to PDE, enter "?ret".

Error Handling:

If you make an error while in LISP, it is sometimes possible to recover by entering control-D. Some errors force you to return to the top level of LISP, which kills PDE. It would have been possible to prevent this by imbedding the break in an errset, but then you could not investigate an error made while in LISP.

Notes:

No notes.

Example 1:

```
pde > b
Break: nil
c{1} (princ "hi")
hit
c{1} ?ret
```

block

Purpose:

To draw the block with no features

Use:

This erases any existing picture, and, if there is a currently active design, draws a picture of the block which is the starting point for the design, but without any features. This command is available so that the user can draw a picture of any subset of the features in a design. Such a picture is created by first calling this command, and then using the "draw" command as often as desired.

Error Handling:

If there is no currently active design, an error message is printed.

Notes:

A. This command is rarely used.

B. This command works whether graphics is on or off.

Example 1:

`pde >` **block**

Three view of a block appear in a new graphics window.

Example 2:

In this example, there is no currently active design.

`pde >` **block**
```
Init_part_display cannot find the block size.
```

c

Purpose:

To change the design.

Use:

The "c" command allows the user to change a feature or the header of the currently active design. It causes PDE to enter into a substantial dialog with the user. The type of dialog varies with the type of feature.

The most complicated parameter to change is "corners", as shown in example 2. Any corner may be changed several times when "corners" is being changed.

The picture of the feature is flashed on and off after the user gives its number so that the user may be sure the correct feature has been selected to change.

This command is not used to insert new features or to delete features, but only to change existing features.

Error Handling:

If an error occurs during a change command, the PDE prompt will reappear. PDE checks first if the feature to be changed exists. If not, a message to this effect will be printed and the PDE prompt will reappear.

Notes:

A. There is no command for changing more than one feature at a time. To change a group or an array of features, either (i) change one at a time or (ii) use the "group" or "array" command to make new features, and then delete the old ones.

B. If you don't know the number of the feature you want to change, use the "pick" command to find out what it is.

Example 1:

The user tries to change a feature that does not exist in the design.

```
pde > c
Enter feature number to be changed or h to change header ? 25
feature does not exist
```

Example 2:

In this example one parameter (the x-value of an endpoint of a straight_groove) is changed.

```
pde > c
Enter feature number to be changed or h to change header ? 5
```

```
1.-  x1
2.-  y1
3.-  x2
4.-  y2
5.-  depth
6.-  width
7.-  bottom_type
8.-  chamfer_in_depth
9.-  reference_feature
```

```
choose a number, 0 to ignore ? 1

 Current value of "x1" is: 6.0
  Enter new value (numeral thru): ? 5.2

More changes for this feature (y/n) ? n
Feature 5 straight_groove is OK.
```

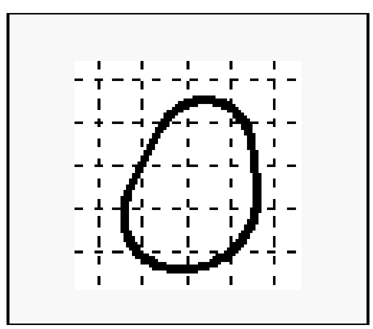
Example 3:

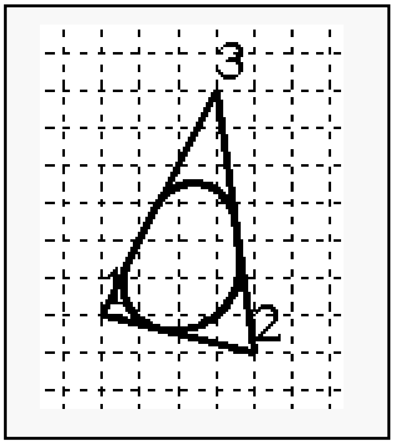
In this example the "corners" parameter of a contour pocket is changed. PDE changes the picture much more frequently than shown here. The contour outline in this example is very simple. It has only three corners. The top view of the contour pocket is shown in four views in this example, as it changes on the screen. The first picture is the original contour pocket.

```
pde > c
Enter feature number to be changed or h to change header ? 4
```

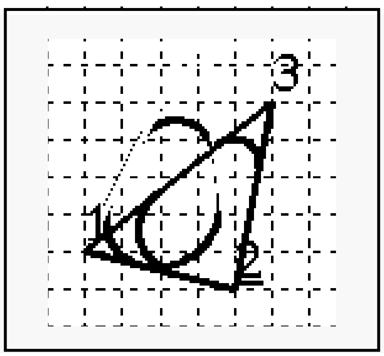
```
1.-  corners
2.-  depth
3.-  reference_feature
```

```
choose a number, 0 to ignore ? 1
```





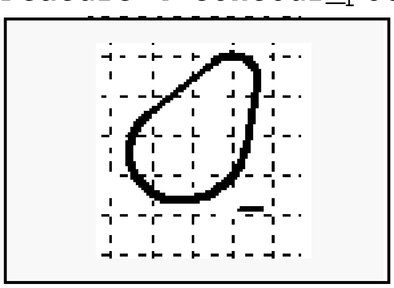
Select a point (numeral) ? **3**
 Current x_coordinate for this corner is 1.75
Enter new x_coordinate ? **2.25**
 Current y_coordinate for this corner is 2.5
Enter new y_coordinate ? **2**
 Current radius for this corner is 0.27
Enter new radius (numeral join_ahead join_back) ? **.165**



Is this correct (y/n) ? **y**

More corner changes (y/n) ? ***maybe***
Enter y or n only
More corner changes (y/n) ? **n**

More changes for this feature (y/n) ? **n**
Feature 4 contour_pocket is OK.



Example 4:

In this example the header is changed. When the block size in the header is changed, the picture of the design will be completely redrawn. That is not shown here.

```
pde > c
Enter feature number to be changed or h to change header ? h
```

```
    1.-  design_id
    2.-  material
    3.-  block_size
    4.-  description
    5.-  add a new attribute
    6.-  delete an attribute
```

```
choose a number, 0 to ignore ? 3

 Current value of "length" is: 6.95
 Enter new value ? 6.2

 Current value of "width" is: 2.975
 Enter new value ? 2.975

 Current value of "height" is: 0.735
 Enter new value ? .97
```

<u>clp</u>

Purpose:

To delete all designs from the PART_DESIGN_LIST and to delete the designs from active memory. If there is a currently active design but no header or features in the design, the command also eliminates the currently active design.

Use:

Just give the command. Be aware that all designs known to PDE will be wiped out of active memory. There is rarely a need to use this command.

Error Handling:

No error handling.

Notes:

A. This command has no effect on stored design files or on the part design index file, part_designs.ndx.

B. Normally, any design handled by PDE since PDE was started will be on the PART_DESIGN_LIST.

C. The "ls" command may be used to see what is currently on the PART_DESIGN_LIST.

D. The "dpd" command may be used to delete a single design.

Example 1:

In this example we list the currently active designs, then use "clp" to delete them. We list the designs again and find that now there are none. We try to print the current design and find that there is none.

```
pde > ls
   junk         junk
 * demo         pde manual demo

pde > clp

Done

pde > ls
Empty

pde> p
no current part_design
```

cls

Purpose:

To clear the command window and return the cursor to the top of that window.

Use:

The command is used if the user wants to be sure to get all of a dialog visible in the command window. It might be used in conjunction with resizing the command window.

Error Handling:

No error handling.

Notes:

This command has no effect on the PDE text window or the graphics window.

Example 1:

`pde >` **cls**

d

Purpose:

To delete a feature.

Use:

This prompts the user to enter a feature number. If the number is the number of a feature in the currently active design, that feature is deleted from the design (and from the picture of the design, if there is one).

Before deleting the feature, the picture of the feature is flashed a few times (if graphics is on), and the user is asked to confirm that the feature should be deleted.

If one or more other features have the selected feature as a reference feature, the user is warned.

Error Handling:

If the feature is not in the design, the user is notified, and no change is made in the design.

Notes:

A. Deleting a feature from the middle of a design leaves a gap in the numbering of the features. Use "rseq" to resequence the feature numbers and eliminate the gap.

Example 1:

```
pde > d
Enter feature number to be deleted ? 44

Feature 44 does not exist.
```

Example 2:

```
pde > d
Enter feature number to be deleted ? 3

Do you want to delete (y/n) ? ok
Enter y or n only
Do you want to delete (y/n) ? y
```

Example 3:

```
pde > d
Enter feature number to be deleted ? 17

Warning: feature 17 is referenced by feature (12 14).
Deletion of feature 17 will force the affected
features to have no reference_feature.

Do you want to delete (y/n) ? n
  No change
```

<u>dpd</u>

Purpose:

To delete a design from active memory and from the PART_DESIGN_LIST.

Use:

This prompts the user to select a specific design to delete from the PART_DESIGN_LIST. The selected design is deleted from the list. If the selected design is currently active, it is deactivated. The selected design is not removed from disk storage if it has been stored there.

If there are no designs on the PART_DESIGN_LIST, PDE prints a message and changes nothing.

Error Handling:

If the user tries to select a design not on the PART_DESIGN_LIST, PDE tells the user to try again.

Notes:

A. This command has no effect on stored design files.

B. The "clp" command may be used to delete all designs on the PART_DESIGN_LIST.

Example 1:

In this example we list the active designs, then delete the design "junk", then list the designs again to see that "junk" has been deleted.

```
pde > ls
   demo         pde manual demo
   darpa1       demo part
 * junk         junk

pde > dpd

    1.-  demo
    2.-  darpa1
    3.-  junk

choose a number, 0 to ignore ? 5
illegal selection - try again? 3

Done

pde > ls
   demo         pde manual demo
   darpa1       demo part
```

<u>draw</u>

Purpose:

To draw a feature.

Use:

If there is a picture, this command prompts the user to enter the number of the feature to be drawn and draws the picture of that feature. If verification is on, the feature is verified first. This command does nothing except print a message if there is no picture on the screen.

The "draw" command is useful in the following situation. Suppose feature A has feature B as its reference feature and feature B is changed. After the change, feature A may not be drawn correctly and may not pass verification. The "draw" command will reverify feature A and redraw it correctly.

Error Handling:

If the feature does not exist, or if graphics is off, PDE notifies the user and does nothing else.

Notes:

A. The "rdraw" command is closely related but serves a different purpose.

B. The "draw" command does not remask the picture.

Example 1:

In this example we call "draw" when there is no picture to draw on.

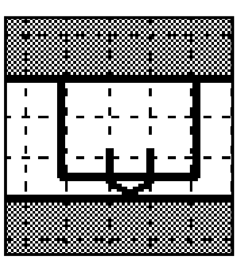
```
pde > draw
not in graphic mode
```

Example 2:

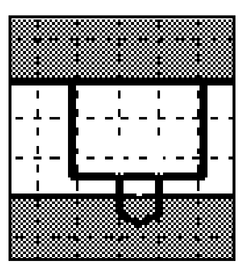
This example shows the situation described under "Use" above.

```
pde > draw

Enter a feature number to draw ? 3
Feature 3 hole is OK.
```

BEFORE



AFTER



e

Purpose:

To select a design to edit

Use:

This displays the entries on the PART_DESIGN_LIST and asks the user to choose one to edit. The chosen design is made to be the currently active design. If graphics is on, the chosen design is drawn.

If the user chooses 0, the command is aborted.

Error Handling:

If the user tries to select an unlisted design, PDE prompts again.

Notes:

A. To create a new design, use the "new" command.

B. If an existing design is on file but is not on the PART_DESIGN_LIST, use the "l" command to load it. It will become the active design.

Example 1:

In this example, the user tries to edit a non-existent design, and then aborts the command.

```
pde > e
    1.-  demo
    2.-  darpa1
    3.-  junk

choose a number, 0 to ignore ? 44
illegal selection - try again?0
abort
```

Example 2:

```
pde > e
    1.-  demo
    2.-  darpa1
    3.-  junk

choose a number, 0 to ignore ? 3
Feature 1 groove is OK.
Feature 2 contour_pocket is OK.
Feature 3 hole is OK.
```

elem

Purpose:

To print a list of the types of features available in PDE.

Use:

This just reminds the user what feature types are available. The list is printed in the PDE text window. Since the same list is presented whenever the "i" command is used, the need to use "elem" is rare.

Error Handling:

No error handling.

Notes:

No notes.

Example 1:

`pde >` **elem**

```
DESIGN FEATURES LIST
--------------------

   1  groove
   2  straight_groove
   3  contour_groove
   4  pocket_corners
   5  pocket_center
   6  contour_pocket
   7  side_contour
   8  text
   9  hole
   10 chamfer_out
```

<u>feat</u>

Purpose:

To display the text of features in a design. This command does not affect the graphics window.

Use:

This prompts the user for a feature number and prints four features of the current design in the PDE text window, starting with the feature number entered by the user.

Error Handling:

If a non-number is entered, the user is prompted again. If the number is larger than the largest feature number or has a decimal point in it, the PDE text window goes blank.

Notes:

If the user enters a number at the PDE prompt, that is the same as calling the "feat" command and then entering the number. Both of the following examples cause the same thing to appear in the PDE text window, as shown in the box below.

Example 1:

```
pde > feat
Enter feature number ? why
Enter feature number ? 3
```

Example 2:

```
pde > 3
```

```
feature  3 - hole
       center_x = 0.5
       center_y = 2.5
      .
      .
      .
feature  4 - hole
       center_x = 2.5
       center_y = 2.5
      .
      .
      .
feature  5 - contour_pocket
      .
      .
      .
feature  6 - groove
       upper_l_x = 0.75
       upper_l_y = 2.25
       lower_r_x = 2.25
       lower_r_y = 0.75
      .
      .
      .
```

<u>flash</u>

Purpose:

To flash a feature.

Use:

This command prompts the user to enter the number of the feature that should be flashed. If a picture of that feature is on the screen, the picture is turned off and on twice. This command is used to identify the picture of a given feature. It is called if the user wants to do something with a feature (such as change, delete, make an array), and has an idea what the number of the feature is, but is not sure. It may also be used to see exactly what is in the picture of a feature.

Error Handling:

If there is no picture of the feature, an error message is printed. If graphics is on but the feature does not exist, a different error message is printed. If a non-number is entered, the user is prompted again.

Notes:

The "pick" command has similar uses.

Example 1:

Suppose graphics is off.

```
pde > flash
Enter a feature number ? abc
Enter a feature number ? 3
That feature has not been drawn.
```

Example 2:

Now suppose graphics is on, but the feature does not exist.

```
pde > flash
Enter a feature number ? 27
this feature does not exist
```

Example 3:

Now suppose graphics is on and the feature exists. Then the picture of the feature flashes off and on twice when the following sequence is entered.

```
pde > flash
Enter a feature number ? 3
```

goff

Purpose:

To set the graphics mode to "off" and wipe out the graphics window, if there is one.

Use:

This sets the PDE graphics mode to "off" and makes any existing picture disappear. When the graphics mode is "off", no picture will be drawn if a new design is made to be the currently active design, and commands that otherwise will have some effect on a picture will not have those effects.

It is usually convenient to call "goff" if you have finished editing a design but want to rename it. If it is renamed with graphics on, it will be redrawn entirely, which wastes a lot of time.

Error Handling:

No error handling.

Notes:

The opposite of "goff" is "gon".

Example 1:

```
pde > goff
graphic display is OFF
```

gon

Purpose:

To set the graphics mode to "on".

Use:

This sets the PDE graphics mode to "on". It does not automatically draw a picture of the currently active design; the "p" command will do this once graphics is "on". Other commands which have graphics effects will have those effects enabled when the graphics mode is "on".

When PDE is started up, it is usually a good idea to make "gon" the first command.

Error Handling:

No error handling.

Notes:

A. If "gon" is called when graphics is already on, nothing changes.

B. The opposite of "gon" is "goff".

C. Use "p" after "gon" to make the graphics window appear and have the design drawn.

Example 1:

```
pde > gon
graphic display is ON
```

group

Purpose:

To duplicate a group of features in a new location.

Use:

This allows the user to duplicate a group of features (such as a pocket with a set of holes at the bottom of the pocket) in another location. It causes PDE to enter into a substantial dialog with the user. It is sensible in its treatment of reference features. Any feature may be duplicated in a group, but it makes little sense to repeat the chamfer_out feature.

When "group" is used, the new features are inserted immediately after the largest feature number in the group. After the insertions, the design is resequenced.

Error Handling:

If one of the features in the group to be copied is not in the design, an error message is printed and the user is prompted to enter a group again.

If the group is not enclosed in parentheses, the user is prompted to enter a group again.

If one of the new features fails verification and the user elects not to try to proceed, or an attempt to proceed fails, the design is restored to the way it was before "group" was called, and the picture is regenerated if graphics is on.

Notes:

A. If a feature in a group (call it A) has a reference feature (call it B) which is the group, then the copy of B is the reference feature for the copy of A. If B is not in the group, then B itself is used as the reference feature for the copy of A.

B. The "array" command can also be used to make several features simultaneously.

C. There is no command to change a group.

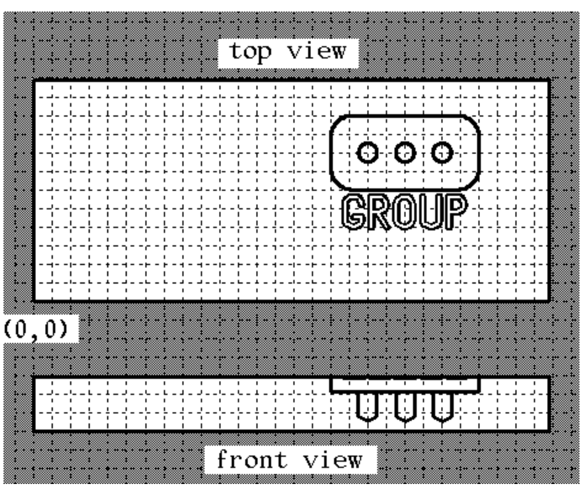
D. Once a group is created, each feature is independent and has no further group affiliation.

Example 1:

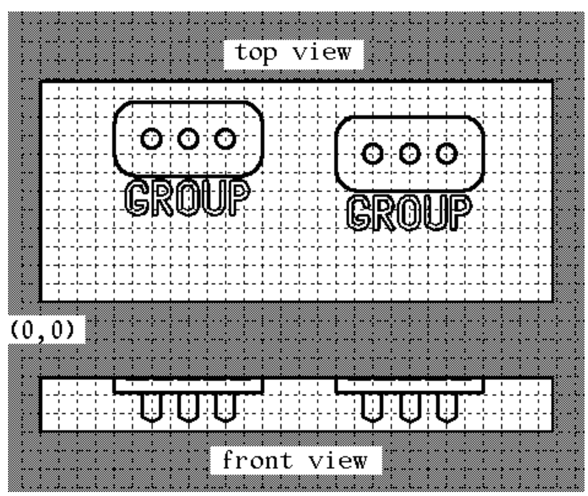
A group of five features is copied from one part of the design to another. The effect of the command on the picture of the design is shown at the bottom of the page. Five features are added to the design itself.

```
pde > group
Enter a list of features to be copied ? (1 2 7)
Feature 7 does not exist.
Enter a list of features to be copied ? 1 2 3 4 5
Enter a list of features to be copied ? (1 2 3 4 5)
Enter x-displacement ? -3
Enter y-displacement ? 0.2
Feature i6 pocket is OK.
Feature i7 text is OK.
Feature i8 hole is OK.
Feature i9 hole is OK.
Feature i10 hole is OK.

Done
```

BEFORE



AFTER



i

Purpose:

To insert a new feature in the currently active design.

Use:

The "i" command enters into a substantial dialog with the user. There is a different dialog for each feature type.

The user is asked for the number of an existing feature before which the new feature should be inserted. To insert a feature at the end, enter any number higher than the highest existing feature number but less than 1024 (it is convenient to use 999 to insert at the end). After a feature is inserted, the design is always resequenced as described for the "rseq" command.

In order to keep similar or nearby features together in a design, it is often handy to insert a feature in the middle of a design.

Error Handling:

If a feature fails verification when the verification mode is "on_soft", the user may enter "go", which tells PDE to save the feature with the error in it. The drawing system may not be able to draw the feature, however. It is usually a good idea to enter "go" and see what happens, but then to use "c" immediately afterward to correct the error.

The dialog for each feature type has its own error checking procedures for individual parameter entries, in addition to the feature verification performed after all the parameters are entered.

Notes:

A. The example given below shows an insert dialog for a pocket. Insert commands are shown in the tutorial at the end of this manual for the features: chamfer, contour_pocket, groove, hole, straight_groove, and text.

B. Features may also be created with the "array" and "group" commands.

Example 1:

pde > **i**
Enter number of existing feature before which to insert
new feature, or 999 to insert at end. ? **2**

The features menu appears in the PDE text window. The user decides to make a pocket by choosing item 4 from the features menu. The user makes four mistakes entering data, but PDE catches three of them immediately.

choose a number, 0 to ignore ? **4**
Enter "upper_l_x" (numeral thru) ? **3**
Enter "upper_l_y" (numeral thru) ? **2.8**
Enter "lower_r_x" (numeral thru) ? ***1***
Enter "lower_r_y" (numeral thru) ? ***thur***
(numeral thru) ? **thru**
Enter "depth" (numeral thru) ? ***,2***
comma not inside a backquote.
(numeral thru) ? **.2**
Enter "corner_radius" (numeral) ? **.33**
Enter "chamfer_in_depth" (numeral/d(efault)=0.046875/n) ? ***yes***
Enter "chamfer_in_depth" (numeral/d(efault)=0.046875/n) ? **d**
Enter "reference_feature" (numeral/n) ? **n**

This completes the dialog for a pocket, but the user has put the right side of the pocket to the left of the left side, so that the length is negative. The verification system discovers this.

The following rule has been broken.
The length of the pocket should be greater than (the
corner radius of the pocket times 2.0) minus 0.00001.
Feature i2 pocket fails verification.
To attempt to continue at risk of further error, enter "go".
Otherwise hit any key and a return.

The user decides to go ahead.

==> **go**

A very peculiar picture is drawn. The new feature is feature number 2, since it was inserted before the old feature 2. The user calls "c" and changes the value of lower_r_x to 5.

l

Purpose:

To load a design file into PDE to be edited or displayed.

Use:

The "l" command prompts the user to enter a design name and checks that the design (with the .pd suffix added) is in the "design" subdirectory. If it is there, the design is loaded, added to the PART_DESIGN_LIST and made to be the currently active design. The first four features of the design are printed in the PDE text window.

Error Handling:

If the design file is not in the "design" subdirectory, an error message is printed.

Notes:

Use "lff" to load a group of designs whose names were stored in a previous PDE session.

Example 1:

```
pde > l
Enter a part design file name ? erewhon
file does not exist
```

Example 2:

```
pde > l
Enter a part design file name ? clamp
;; Loading file "./design/clamp.pd"
```

lff

Purpose:

To load all the designs in the file "part_designs.ndx" from the "design" subdirectory into the LISP environment and add them to the PART_DESIGN_LIST.

Use:

The "lff" command may be used just after starting PDE to restore the PDE environment to the way it was in a previous session, if the "stf" command was used near the end of that session.

Error Handling:

If the part_designs.ndx file does not exist, an error message is printed. If one of the files to be loaded by the part_designs.ndx file does not exist, an error message is printed, and none of the rest of the files named in the part_designs.ndx file is loaded.

Notes:

Use "l" to load a single design by name.

Example 1:

In this example, we use "ls" to see what the active designs are. Then we use "stf" to save the names of the designs, get out of PDE, get out of LISP, get back into LISP, and get back into PDE. Finally we use "lff" to restore the designs we were working on.

```
pde > ls
   clamp        clamp
   lok1         locking clevis first cut
 * lok2         locking clevis second cut

pde > stf
./design/part_designs.ndx  saved

pde > q

nil
=> (exit)

23} vws2_lisp
=> (pde)

pde > lff
;; Loading file "./design/part_designs.ndx"
;; Loading file "./design/clamp.pd"
;; Loading file "./design/lok1.pd"
;; Loading file "./design/lok2.pd"
./design/part_designs.ndx  loaded
```

loc

Purpose:

To find the coordinates (in terms of x, y and depth) of a point on the design.

Use:

If there is a picture, this command prompts the user to use the mouse to point at the picture and press the right mouse button. When the right button is pressed, a cross hair is displayed at the position of the mouse cursor, and the location of the point picked is shown in the upper right corner of the picture, given in part coordinates. Position information is rounded off to the nearest eightieth of an inch.

The user may then locate another point by moving the mouse and pressing the right button, or may get out of "loc" by pressing the middle or left button. The position information and the cross hair remain on the screen.

When the mouse cursor is moved into the graphics window, it changes shape into a hand with a pointing finger. The point that is picked is the one at the end of the finger.

The "loc" command is useful when the user can visually identify a place on the graphics window where a feature should be located and would like to know its coordinates.

The "loc" command may be called in the middle of a dialog with PDE whenever PDE is asking for some sort of x-value, y-value, or depth.

Error Handling:

If the user presses the wrong mouse button, a message is printed.
If graphics is not on an error message is printed.

Notes:

A. The "loc" command does not automatically enter any data when it is used inside a dialog. After "loc" is finished, the dialog continues where it left off, and the user must enter an appropriate response.

B. If "loc" is active and no button is pressed for 100 seconds, "loc" times out.

C. The "pick" command uses the mouse to identify feature numbers.

Example 1:

In this example, the user tries "loc" when graphics is off.

```
pde > loc
graphic display is OFF
```

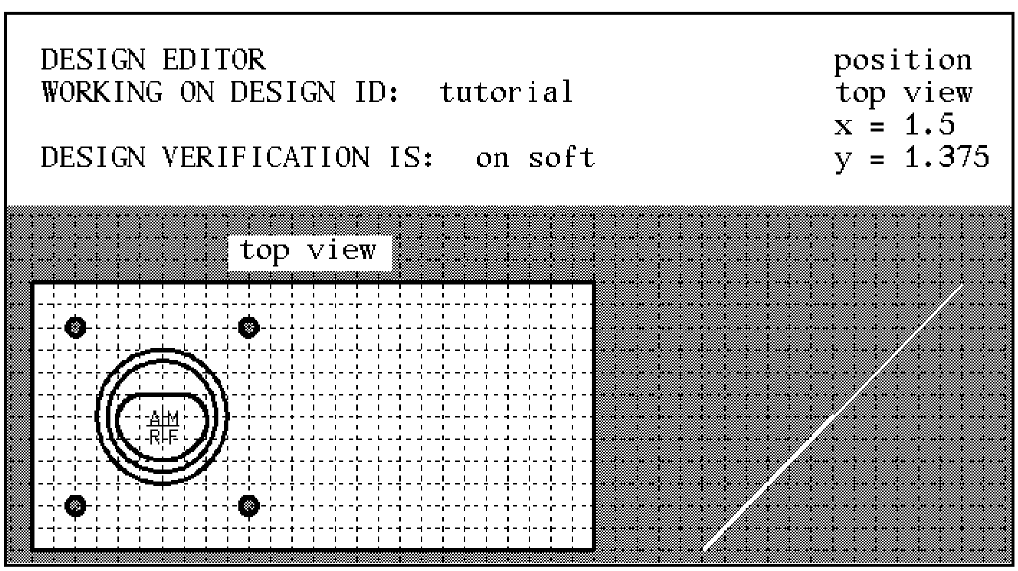
Example 2:

```
pde > loc
Point at the picture and press the right mouse button.
```
The user presses the wrong button.
```
That was the middle button.
Point at the picture and press the right mouse button.
```
The user points and presses the right button. A cross hair with the letters AMRF appears as shown in the picture below, and the position information is printed in the upper corner of the picture.
```
Press right button to relocate, left or middle to return.
```
The user presses the left button to get out of "loc".

Example 3:

In this example, "loc" is called in the middle of a feature changing operation.

```
pde > c
Enter feature number to be changed or h to change header ? 1
choose a number, 0 to ignore ? 1

 Current value of "center_x" is: 0.5
 Enter new value (numeral): ? loc
Point at the picture and press the right mouse button.
```
The user points and presses the right button. A cross hair with the letters AMRF appears as shown in the picture below, and the position information is printed in the upper corner of the picture.
```
Press right button to relocate, left or middle to return.
```
The user presses the left button to get out of "loc" and gets another prompt from the change routine.



ls

Purpose:

To list the design_id's and the descriptions of all the designs on the PART_DESIGN_LIST.

Use:

This command shows the user which designs are currently active and can be selected for editing using the "e" command.

The listing is displayed in the PDE command window.

Error Handling:

No error handling.

Notes:

A. The PART_DESIGN_LIST exists only while PDE is in operation, and includes those designs which have been called up for editing and not deleted from the list with "dpd" or "clp".

B. A separate list of design names is kept in a file called part_designs.ndx in the design subdirectory. The "lff" command will load all the designs in that file, and after loading they will appear on the PART_DESIGN_LIST.

C. If you want to see what designs are actually in the design subdirectory, use UNIX to list the contents of the subdirectory.

Example 1:

`pde >` **ls**

```
    fluidamp    FLUIDIC AMPLIFIER
    onr         onr demo
  * tutorial    tutorial part
    junk        junk
```

m

Purpose:

To print a list of PDE commands. "m" is short for "menu".

Use:

This displays the main menu in the PDE text window. It is used when the user wants to be reminded what commands are available.

Error Handling:

No error handling.

Notes:

A. The menu is arranged in three groups of related commands.

B. The menu is displayed automatically when PDE starts up.

Example 1.

pde > **m**

This causes the menu shown at the right to appear in the PDE text window.

```
      PART DESIGN EDITOR COMMANDS

clp    Clear All Designs
dpd    Delete a Design
e      Select a Design to Edit
l      Load a Design File
lff    Load Designs from Index File
ls     List Designs
n      Rename a Design
new    Generate a New Design
p      Display the Current Design
save   Save the Current Design to Disk
feat   Display Design Features
stf    Store Designs to Index File
c      Change the Design
d      Delete a Feature
i      Insert a Feature
array  Make a Feature Pattern
group  Duplicate a Group of Features
rseq   Resequence Design Features

           SYSTEM COMMANDS

b      Break
cls    Clear this Screen
elem   Print a List of Features
m      Print a List of Commands
q      Quit from Part Design Editor

  GRAPHICS AND VERIFICATION COMMANDS

block  Draw the Block with no Features
draw   Draw a Feature
flash  Flash a Feature
goff   Set the Graphic Mode to OFF
gon    Set the Graphic Mode to ON
loc    Use Mouse to Pick a Position
pick   Use Mouse to Pick a Feature
rdraw  Redraw the Screen
verif  Verify a Design
vset   Set the Verification Mode
```

n

Purpose:

To rename a design

Use:

This copies the current design into a new data structure and prompts the user for a new design_id and new description. The new design is made the currently active design. If graphics is on, a new picture is drawn.

The "n" command is useful for copying a design to be modified.

Error Handling:

If there is already a design on the PART_DESIGN_LIST with the name chosen for the new design, the "n" command is aborted.

Notes:

A. Use the "new" command to create a design from scratch.

Example 1:

Suppose "gnu_design" already exists.

```
pde > n
Enter new part_design name ? gnu_design

part_design name:  gnu_design  already exists - ignored
```

Example 2:

Suppose a design named "knew_her_design" is active. The following dialog creates a new design named "knuth_design", with the same size block, same material, and same features as the "knew_her_design".

```
pde > n
Enter new part_design name ? knuth_design
Enter new design description ? most new design
```

<u>new</u>

Purpose:

To create a new design.

Use:

This allows the user to generate a new design from scratch. The user is prompted for the name of the design, a brief description (not more than 30 characters), the size of the block, and (optionally) the material to be used. The user may add, delete, or change features while still conducting a dialog with the "new" command. Once the user gets out of the "new" command, a PDE prompt is presented.

Error Handling:

The user will be prompted again if the user enters anything other than a number for the length, width, and height of the block, an integer from 0 to 5 when choosing a material from the menu, or an integer from 0 to 3 when asked to decide to add, delete, or change a feature.

Error handling while inserting, deleting, or changing a feature is identical to that for the "i", "d", and "c" commands, respectively.

Notes:

A. PDE will not object if a new design is created with the same name as an existing design. When the user calls "save" to write the design into a file, PDE will warn the user if the file exists.

B. The "n" command is used to rename a design, not to make a new one.

Example 1:

```
pde > new
Enter the design_id ? gnu_knew_design

gnu_knew_design  created
Enter part design description ? new demo part

use default block_size (y/n) ? n
Enter block_length ? yes
Enter block_length ? 6.25
Enter block_width ? 3.0
Enter block_height ? .75
```

```
1.-  aluminum
2.-  brass
3.-  steel
4.-  monel
5.-  not_specified
```

```
choose a number, 0 to ignore ? 8
choose a number, 0 to ignore ? 2
```

The graphics window appears, and a picture of the block with no features is drawn.

```
Enter feature 1
```

The features menu is shown in the PDE text window. The user decides to make a hole (9).

```
choose a number, 0 to ignore ? 9
```

A dialog like the dialog for inserting a feature (the "i" command) follows.

```
Feature 1 hole is OK.
Enter 1 = add a feature, 2 = delete a feature,
3 = change a feature, or 0 to ignore ? 2
```

A dialog like the dialog for deleting a feature (the "d" command) follows.

```
Enter 1 = add a feature, 2 = delete a feature,
3 = change a feature, or 0 to ignore ? 0

pde >
```

p

Purpose:

To display the text of the current design in the PDE text window, and, if graphics is on, to draw a picture of the design.

Use:

The "p" command may be useful when an editing session has proceeded with graphics off, and then graphics is turned on. Even when graphics is turned on, no picture will appear until "p" is called.

This prints the first four features of the current design in the PDE text window. If graphics is on and there is an existing picture, it is removed and the current design is drawn.

Error Handling:

No error handling.

Notes:

A. It is usually not a good idea to use "p" to clean up an existing picture, because "p" regenerates the image of each feature. The "rdraw" command is better for that purpose.

B. The "feat" command, or simply typing a feature number after the PDE prompt, will also display feature descriptions in the PDE text window.

Example 1:

`pde >` **p**

<u>pick</u>

Purpose:

To find the feature number of a feature showing in the graphics window.

Use:

If there is a picture, this command prompts the user to use the mouse to point at a feature on the picture and press the left button. When the left button is pressed, the number of the feature is displayed on the command screen. If the pick fails, a message is printed in the command screen.

The pick command may be used to identify a feature for the purpose of deleting it, changing it, making an array, etc.

When the mouse cursor is moved into the graphics window, it changes shape into a hand with a pointing finger. The point that is picked is the one at the end of the finger. This point must be on one of the black lines outlining the feature in any of the three views of the feature.

Error Handling:

If "pick" is called when graphics is off or when graphics is on but there is no picture, an error message is printed.

Notes:

A. The "loc" command uses the mouse to find the coordinates of a point on the design.

Example 1:

In this example, "pick" is called when there is no picture to pick on.

pde > **pick**
```
No features have been drawn to pick.
```

Example 2:

Suppose there is a picture of a design with features on it, but the user misses the mark.

pde > **pick**
```
Point at the feature and press the left mouse button.
No feature picked.
```

Example 3:

Suppose there is a picture of a design with features on it, and the user picks successfully.

pde > **pick**
```
Point at the feature and press the left mouse button.
Feature 2
```

q

Purpose:

To quit from the Part Design Editor.

Use:

This gets the user out of the Part Design Editor and back to wherever the editor was called from (usually either LISP or vws_cadm). Since the editor is no longer running, there is no prompt from the editor. If the editor has drawn a picture, it will disappear. The command window is cleared, but the text window is not cleared.

Error Handling:

No error handling.

Notes:

A. The user is not prompted about any designs which have been edited but not saved, so it is a good idea to think twice before using "q".

B. LISP can grow to five Mb. When PDE is operated for over an hour with complex graphics, or for several hours with simple graphics, LISP will be filled up and stop operating. It is a good idea to stop work with "q" at least every two hours, get out of vws2_lisp, and restart vws2_lisp. It takes two to five minutes to get out of PDE, get back in, and reload a design.

Example 1:

`pde >` **q**

The command window clears and the cursor goes to the top.

```
nil
=>
```

rdraw

Purpose:

To redraw the graphics window and regenerate the mask.

Use:

This redisplays the picture, if there is one. This command produces a better picture than if the picture is redisplayed via suntools, since "rdraw" redisplays pictures of features and masks in the correct order. This command also deletes any pictures of fonts and any position information that may have been drawn during the process of making a text feature or using the "loc" command.

Because "rdraw" simply turns existing images off and on (except for the grey background mask, which is regenerated), it is much faster than a call to "p", which regenerates all the images. Using "rdraw" helps the user get a better idea of what the part really looks like, since a truer representation results from the improved masking.

The "rdraw" command is often used just before making a screen dump for saving the picture of a design.

Error Handling:

If "rdraw" is called with graphics on but no picture showing, an error message is printed.

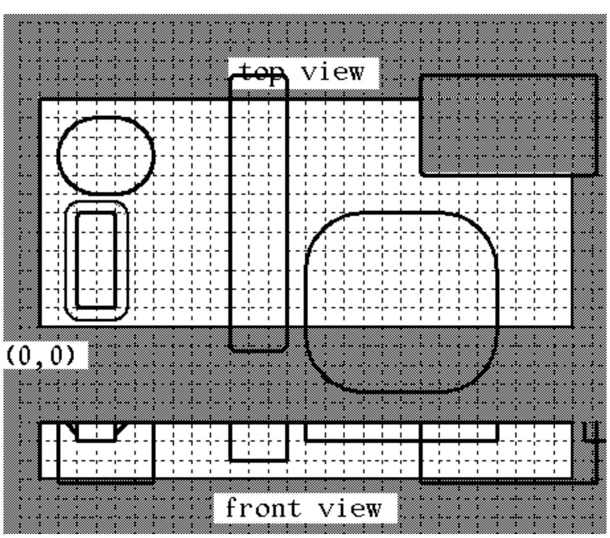
Notes:

No notes.

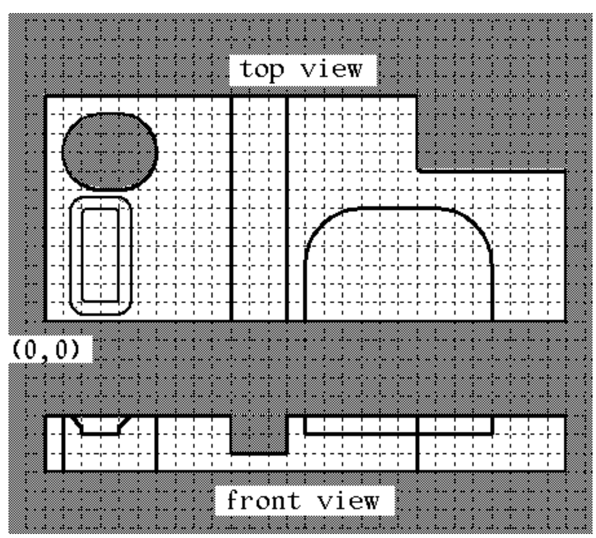
Example 1:

`pde >` **rdraw**

BEFORE



AFTER



rseq

Purpose:

To renumber the features of a design so that they are in numerical order, starting with 1, with no gaps.

Use:

If features are deleted from a design, gaps in the sequence of feature numbers may appear. This command resequences the feature numbers so that they once again are 1, 2, 3, etc. In doing this, it often has to change reference feature numbers, as well.

Error Handling:

If "rseq" is called when no design is active, an error message is printed.

Notes:

It is not necessary to resequence after "i" (insert), "array" or "group", since resequencing is done automatically for those commands.

Example 1:

`pde >` **rseq**

In this example, a design has only two features (numbered 1 and 3), and one is a reference feature for the other. The call to "rseq" changes the numbers to 1 and 2, and adjusts the reference feature numbering properly.

BEFORE

```
feature  1 - pocket_corners
       reference_feature = 3
       upper_l_x = 6
       upper_l_y = 5.5
       lower_r_x = 8
       lower_r_y = 3.5
       depth = thru
       corner_radius = 1

 feature  3 - pocket_corners
       upper_l_x = 5
       upper_l_y = thru
       lower_r_x = 10
       lower_r_y = thru
       depth = 1
       corner_radius = 0.2
```

AFTER

```
feature  1 - pocket_corners
       reference_feature = 2
       upper_l_x = 6
       upper_l_y = 5.5
       lower_r_x = 8
       lower_r_y = 3.5
       depth = thru
       corner_radius = 1

 feature  2 - pocket_corners
       upper_l_x = 5
       upper_l_y = thru
       lower_r_x = 10
       lower_r_y = thru
       depth = 1
       corner_radius = 0.2
```

save

Purpose:

To save the current design in a file.

Use:

When "save" is called, two versions of the current design are written into files (in disk storage). One is very easy for a human to read. Its name is the design id with the suffix ".prt". The other is LISP readable (and fairly easy for a human to read). Its name is the design id with the suffix ".pd". Both files are created in the "design" subdirectory.

If the design is already on file, the user is asked if the old version should be overwritten.

Error Handling:

No error handling. If the "design" subdirectory does not exist, the files will not be written.

Notes:

If there is no currently active design, files named nil.pd and nil.prt are written.

Example 1.

```
pde > save
./design/dog.pd exists
OK to overwrite (y/n) ? y
./design/dog.pd  saved
./design/dog.prt  saved
```

stf

Purpose:

To write a file which may be used to load all the designs which are currently active in this session with PDE when PDE is used another time.

Use:

This writes a file named part_designs.ndx in the design subdirectory so that the file will cause only the designs on the current PART_DESIGN_LIST to be loaded. If there is an existing file with that name, it is overwritten.

Error Handling:

No error handling.

Notes:

A. This command is not used frequently, since it is simple to load any design with the "l" command, and there is rarely a need to have several designs active simultaneously.

B. The "lff" command loads the designs listed in part_designs.ndx into PDE.

Example 1:

```
pde > stf
./design/part_designs.ndx   saved
```

verif

Purpose:

To verify a design.

Use:

The entire currently active design is verified. The user is notified feature by feature whether the feature passes or fails verification. If any feature fails, a list of all the features that failed is presented to the user at the end.

If a user wants to be sure that a design is OK, "verif" may be called just before the design is saved.

Error Handling:

If there is no active design when "verif" is called, a message is printed that says that.

Notes:

It does not matter how the verification mode is set when verif is called.

Example 1:

The "verif" command is called to verify a design with two features.

```
pde > verif
Verifying the design: junk
Feature 1 groove is OK.
Feature 2 contour_pocket is OK.
Entire design is OK.
```

Example 2:

The "verif" command is called when there is no active design.

```
pde > verif
There is no active design.
```

vset

Purpose:

To set the verification mode.

Use:

A message identifying the current verification level is printed. The user is prompted to choose one of the three verification modes, "off", "on_soft", or "on_hard", from a menu in the PDE text window. The PDE verification mode is set to the chosen level.

If the user enters "0", the verification mode is not reset.

Error Handling:

If the user enters anything other than 0, 1, 2, or 3, an error message is printed and the user is prompted again.

Notes:

A. When PDE starts up, the verification mode is set to "on_soft". It is almost always best to leave it that way.

B. It is OK to set the mode to what it already is.

Example 1:

`pde >` **vset**
The menu shown at the right appears in the PDE text window.
```
Current verify_flag is "on_soft"
choose a number, 0 to ignore ? 0
abort
```

```
1. - off
2. - on_soft
3. - on_hard
```

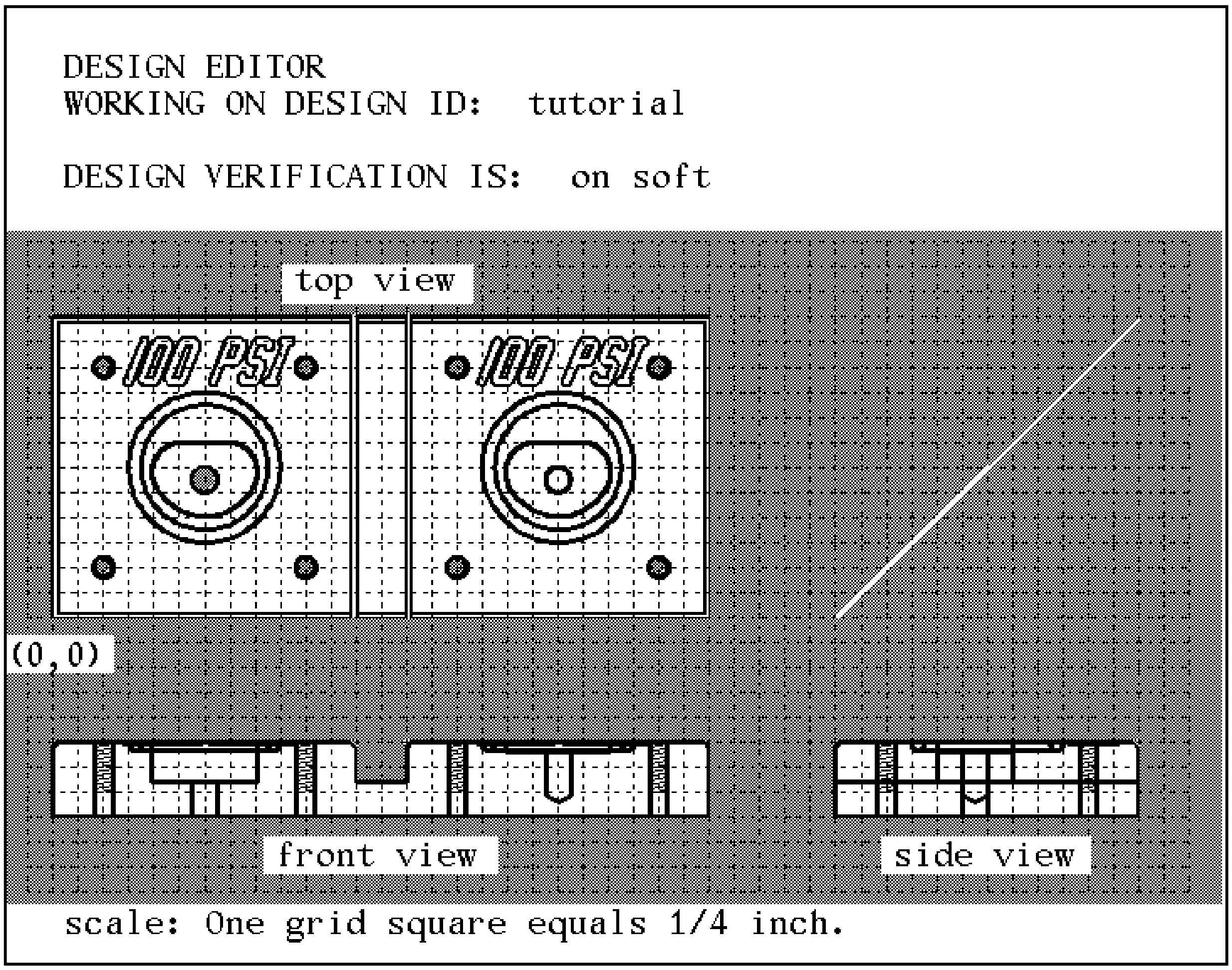
Example 2:

`pde >` **vset**
The menu shown at the right appears in the PDE text window.
```
Current verify_flag is "on_soft"
choose a number, 0 to ignore ? 5
illegal selection - try again? 1

verify_flag is off
```

III. PART DESIGN EDITOR TUTORIAL

This is a tutorial for you to use to train yourself in using PDE. The convention regarding the use of fonts followed in this tutorial was described on page B-3 of this manual. To use the tutorial, log in to your Sun and be sure your environment is set up as described on page B-2 of the manual. Do not start a window system yet.

This tutorial will have you design the part shown in the picture immediately below.



First, enter the following on the terminal (which should be using the full screen at this point).

suntools pde_windows

This should format your screen with a set of windows. Put the mouse cursor in the window labelled LISP and enter the following:

vws2_lisp

In a few seconds, LISP should start up and show you an arrow prompt. From here to the end of the tutorial, enter the text shown in **boldface Times font** in your command screen. All the following material shown in `courier font` should be printed on your screen by the system. Explanatory material is in this kind of font. Sometimes you will be directed to use the mouse. If you make an error, read about the command in which you made the error in the commands section of this manual to try to recover. Good luck.

=> **(pde)**

The command screen clears, and we start by turning graphics on and defining the size of the block.

`pde >` **gon**
`graphic display is ON`

`pde >` **new**
`Enter the design_id ?` **tutorial**

`tutorial  created`
`Enter part design description ?` **tutorial part**

`use default block_size (y/n) ?` **n**
`Enter block_length ?` **6.5**
`Enter block_width ?` **3.0**
`Enter block_height ?` **.75**

```
1.-  aluminum
2.-  brass
3.-  steel
4.-  monel
5.-  not_specified
```

`choose a number, 0 to ignore ?` **2**

The graphics screen formats itself, showing three views of the block with no features.

```
Enter feature 1
```

```
1.- groove
2.- straight_groove
3.- contour_groove
4.- pocket_corners
5.- pocket_center
6.- contour_pocket
7.- side_contour
8.- text
9.- hole
10.- chamfer_out
```

The menu in the box is the "features menu". It appears frequently on the screen, but we will not show it again in this tutorial. We now create the hole on the lower left of the block. It will be countersunk and threaded.

```
choose a number, 0 to ignore ? 9
Enter "center_x" (numeral) ? .5
Enter "center_y" (numeral) ? .5
Enter "diameter" (numeral) ? .1719
Enter "depth" (numeral/thru) ? thru

Adding threads to this hole (y/n) ? y
Enter "thread_diameter" (numeral) ? .19
Enter "thread_depth" (numeral thru) ? .5
Choose "threads_per_inch" (8 12 16 20 24 32) ? 24
Enter "countersink_diameter" (numeral/n) ? .2
Enter "reference_feature" (numeral/n) ? n
Feature 1 hole is OK.
```

The hole is drawn on the picture and the following feature description is printed in the PDE text window. PDE prints a lot of feature descriptions in the text window, but we will mention only a few of them in the rest of this tutorial.

```
feature  1 - hole
    center_x = 0.5
    center_y = 0.5
    diameter = 0.1719
    depth = thru
    thread_diameter = 0.19
    thread_depth = 0.5
    threads_per_inch = 24
    countersink_diameter = 0.2
```

```
Enter 1 = add a feature, 2 = delete a feature, 3 = change a feature,
or 0 to ignore ? 1

Enter feature 2
```

Features menu appears in PDE text window. The next feature we make is a contour pocket. A lot of drawing goes on in the graphics window that is not mentioned below.

```
choose a number, 0 to ignore ? 6

Enter x, y coordinate for point 1
Enter x_coordinate ? 1.5
Enter y_coordinate ? 1.5
Is this correct (y/n/b) ? y
more points (y/n) ? y

Enter x, y coordinate for point 2
Enter x_coordinate ? 2
Enter y_coordinate ? 1
Is this correct (y/n/b) ? b      Here we decide to redo corners 1 and 2.
More backup (y/n) ? y            The "b" option is short for "back up".
More backup (y/n) ? n
more points (y/n) ? y

Enter x, y coordinate for point 1
Enter x_coordinate ? .8
Enter y_coordinate ? 1.75
Is this correct (y/n/b) ? y
more points (y/n) ? y

Enter x, y coordinate for point 2
Enter x_coordinate ? 1.5
Enter y_coordinate ? .3
Is this correct (y/n/b) ? y
```

```
more points (y/n) ? y

Enter x, y coordinate for point 3
Enter x_coordinate ? 2.2
Enter y_coordinate ? 1.75
Is this correct (y/n/b) ? y
more points (y/n) ? n

 Specify a radius at each corner:

Select a point (numeral/done) ? 1
Current radius for this corner is 0
Enter a new radius (numeral/join_ahead/join_back) ? .3

Is this correct (y/n) ? y
Select a point (numeral/done) ? 2
Current radius for this corner is 0
Enter a new radius (numeral/join_ahead/join_back) ? .3

Is this correct (y/n) ? y
Select a point (numeral/done) ? 3
Current radius for this corner is 0
Enter a new radius (numeral/join_ahead/join_back) ? .3

Is this correct (y/n) ? y
Select a point (numeral/done) ? 2
Current radius for this corner is 0.3
Enter a new radius (numeral/join_ahead/join_back) ? join_back

Is this correct (y/n) ? y
Select a point (numeral/done) ? done
Enter  "depth" (numeral) ? .1
Enter "reference_feature" (numeral/n) ? n
Feature 2 contour_pocket is OK.
```

The contour pocket is drawn on picture. Next, we use the first hole we created to make an array of four holes (the original plus three copies). The "pick" command is used to get the feature number of the first hole, in case we have forgotten it.

```
Enter 1 = add a feature, 2  delete a feature,
      3 = change a feature, or 0 to ignore ? 0

pde > pick
Point at the feature and press the left mouse button.
```

Point at the edge of the hole and press the left mouse button. If the pick fails, try again.

Feature 1

pde > **array**
Enter feature number to be repeated ? **1**

An array of features may be rectangular or in a circular arc.
Choose a type (1=rectangular 2=circular). ? **1**

Array size is given either by the total x and y distances,
or by the incremental x and y distances between features.
Choose a method (1=incremental 2=total). ? **1**

You will also need to specify the number of times the
feature is to be repeated in the x and y directions.

Enter no. of times to be repeated in x-direction ? **2**
Enter no. of times to be repeated in y-direction ? **2**
Enter x-displacement ? **2**
Enter y-displacement ? **2**

Is this correct (y/n) ? **y**

Another hole appears on the picture after each of the following three lines.

Feature i2 hole is OK.
Feature i3 hole is OK.
Feature i4 hole is OK.

Done

pde > **i**
Enter number of existing feature before which to insert
new feature, or 999 to insert at end. ? **5**

The features menu appears in the PDE text window. Next we make a hole in the center of the contour pocket. Since we want the hole at the bottom of the pocket, we use the pocket as the reference feature for the hole. The "loc" command is used to find the coordinates of the center.

choose a number, 0 to ignore ? **9**
Enter "center_x" (numeral) ? **loc**
Point at the picture and press the right mouse button.

Point at the center of the contour pocket and press the right mouse button. If your aim is good, the following four lines will appear at the top right of the graphics window.

```
position
top view
x = 1.5
y = 1.375
```

```
Press right button to relocate, left or middle to return.
```

Press the left button while the mouse cursor is over the picture. PDE will continue with the dialog for creating a hole that it had started when we called "loc".

```
Enter  ? 1.5
Enter "center_y" (numeral) ? 1.375
Enter "diameter" (numeral) ? .25
Enter "depth" (numeral/thru) ? .5
```

```
1. - conical
2. - flat
```

```
choose a number, 0 to ignore ? 1

Adding threads to this hole (y/n) ? n
Enter "countersink_diameter" (numeral/n) ? n
Enter "chamfer_in_depth" (numeral/d(efault)=0.046875/n) ? n
Enter "reference_feature" (numeral/n) ? 5
Feature i5 hole is OK.
```

The hole appears on the picture.

`pde > ` **5** This is equivalent to using the "feat" command.

Look at the PDE text screen. Note that the hole just added is feature 5, since it was added before the old feature 5. Note also that the contour pocket, which was feature 2 originally, is now feature 6. This is because the array of small holes was also added before the contour pocket, just following feature 1.

Look at the front view of the new hole on the picture, and note that the top of the hole is at the bottom of the contour pocket.

```
pde > i
Enter number of existing feature before which to insert
```

```
new feature, or 999 to insert at end. ? 999
```

The features menu appears in the PDE text window. Next we make the round groove around the contour pocket.

```
choose a number, 0 to ignore ? 1
Enter  "upper_l_x" (numeral thru) ? .75
Enter  "upper_l_y" (numeral thru) ? 2.25
Enter  "lower_r_x" (numeral thru) ? 2.25
Enter  "lower_r_y" (numeral thru) ? .75
Enter  "depth" (numeral) ? .08
Enter  "width" (numeral) ? .125
Enter  "corner_radius" (numeral) ? .75
```

```
1. - round
2. - flat
```

```
choose a number, 0 to ignore ? 2
Enter "chamfer_in_depth" (numeral/d(efault)=0.046875/n) ? n
Enter "chamfer_out_depth" (numeral/d(efault)=0.046875/n) ? n
Enter "reference_feature" (numeral/n) ? n
Feature i999 groove is OK.
```

```
pde > i
Enter number of existing feature before which to insert
new feature, or 999 to insert at end. ? 999
```

The features menu appears in the PDE text window. Next we make the words "100 psi".

```
choose a number, 0 to ignore ? 8
Enter "text" (eg. AMRF VWS) ? 100 psi
```

A picture of the five available fonts appears at top right of graphics window.

```
Enter a font number ? 4
Enter "lower_l_x" ? .75
Enter "lower_l_y" ? 2.35
Enter "height" ? .4
Enter "depth" ? .015
```

A list of available text widths appears in the PDE text window.

```
choose a number, 0 to ignore ? 3
Enter "reference_feature" (numeral/n) ? n
```

```
Feature i999 text is OK.
```

The text is drawn on the picture. Next we make the entire right side of the design in one fell swoop by using the "group" command.

```
pde > group
Enter a list of features to be copied ? (1 2 3 4 5 6 7 8)
Enter x-displacement ? 3.5
Enter y-displacement ? 0
```

Another feature is drawn after each of the following eight verification messages.

```
Feature i9 hole is OK.
Feature i10 hole is OK.
Feature i11 hole is OK.
Feature i12 hole is OK.
Feature i13 contour_pocket is OK.
Feature i14 groove is OK.
Feature i15 text is OK.
Feature i16 hole is OK.

Done
```

Now we go back and change the first contour pocket. We think it is feature 6, but we are not sure, so we use "flash" to check. Then we increase the depth of the pocket.

```
pde > flash
Enter a feature number ? 6

pde > c
Enter feature number to be changed or h to change header ? 6
```

```
    1.-  corners
    2.-  depth
    3.-  reference_feature
```

```
choose a number, 0 to ignore ? 2
Current value of "depth" is: 0.1
Enter new value (numeral): ? 0.4

More changes for this feature (y/n) ? n
Feature 6 contour_pocket is OK.
```

The contour pocket is redrawn. Now the hole at the bottom of the pocket is hanging in mid-air in the front and side views of the pocket, so we redraw the hole.

```
pde > draw
Enter a feature number to draw ? 5
Feature 5 hole is OK.

pde > i

Enter number of existing feature before which to insert
new feature, or 999 to insert at end. ? 999
```

The feature menu appears in the PDE text window. We are going to chamfer the block.

```
choose a number, 0 to ignore ? 10
Enter "chamfer_out_depth" (numeral/d(efault)=0.046875/n) ? .04
Feature i999 chamfer_out is OK.
```

A chamfer of the block is drawn on picture. The last feature we add is a straight groove to separate the two halves of the design.

```
pde > i

Enter number of existing feature before which to insert
new feature, or 999 to insert at end. ? 999
```

The features menu appears in the PDE text window. We choose a straight_groove.

```
choose a number, 0 to ignore ? 2
Enter  "direction" (1=horizontal / 2=vertical / 3=oblique) ? 2
Enter  "x1"  (numeral/thru) ? 3.25
Enter  "y1"  (numeral/thru) ? thru
Enter  "y2"  (numeral/thru) ? thru
Enter  "depth" (numeral) ? .4
Enter  "width" (numeral) ? .5

    1. - round
    2. - flat

choose a number, 0 to ignore ? 2
Enter "chamfer_in_depth" (numeral/d(efault)=0.046875/n) ? .04
Enter "reference_feature" (numeral/n) ? n
Feature i999 straight_groove is OK.
```

A picture of the straight groove is drawn. Now we use "rdraw" to improve the picture. The font picture disappears and the picture is redrawn with new masking.

`pde >` **rdraw**

We save the design. If the tutorial has been done in your directory before, PDE will ask you if it is OK to overwrite the existing file. Say yes. That is not shown here.

`pde >` **save**
```
./design/tutorial.pd  saved
./design/tutorial.prt  saved
```

`pde >` **q**

PDE quits, the graphics window disappears, and we are back in LISP in the command window. We get out of LISP as follows.

=> **(exit)**

If you wish, you can look at the design files you just created in the LISP window by entering the UNIX commands:

more design/tutorial.pd
more design/tutorial.prt

REFERENCES

[KR&J]
Kramer, Thomas R.; and Jun, Jau-Shi; "The Design Protocol, Part Design Editor, and Geometry Library of the Vertical Workstation of the Automated Manufacturing Facility at the National Bureau of Standards"; NBSIR 88-3717; 1988; National Bureau of Standards; 101 pages.

[K&S1]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention and Detection in Data Preparation for the Vertical Workstation Milling Machine in the Automated Manufacturing Facility at the National Bureau of Standards"; NBSIR 87-3677; National Bureau of Standards; 1987; 61 pages.

[K&S2]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention in Data Preparation for a Numerically Controlled Milling Machine"; Proceedings of 1987 ASME Annual Meeting; 1987; ASME; PED-Vol. 25, pp. 195 - 213.

[SUN1]
"Getting Started with UNIX: Beginner's Guide"; Sun Microsystems; Part No: 800-1284-03; 1986; 143 pages.

[SUN2]
"Windows and Window Based Tools: Beginner's Guide"; Sun Microsystems; Part No: 800-1287-03; 1986; 186 pages.