

PV-WAVE Foundation/Advantage Quick Start Guide

What is PV-WAVE?

PV-WAVE Foundation is a powerful, interactive command language, complete with all the program control options of a structured programming language, but in a concise, easy-to-learn form. In order to expand capabilities and performance, all of the numerical functions from IMSL's C/Math/Library and C/Stat/Library have been integrated into PV-WAVE to create PV-WAVE *Advantage*. This document is designed to help a new user quickly learn how to use PV-WAVE. Additionally, a Quick Reference Guide is included at the end of this to help you easily locate information on any aspect of PV-WAVE

General Information:

The following list contains basic concepts, rules, and tips that you will use over and over during any PV-WAVE session. More detail about each item is provided later in this document.

- ⇒ Start on-line help by typing "help" at the WAVE prompt. Start on-line documentation by typing "help, /doc" at the WAVE prompt or by typing "wavedoc" at an operating system (OS) prompt after executing your wvsetup file (wvsetup, wvsetup.sh or wvsetup.com).
- ⇒ Obtain on-line information about saved variables, procedures, etc., by entering INFO at the WAVE> prompt.
- ⇒ PV-WAVE is not case sensitive; upper and lower case letters are used in this guide to enable you to learn command names and keywords more easily. Upper case is used for procedure and function names. Lower case is used for variable names. Initial letter in uppercase represents a keyword.
- ⇒ Press the <Return> key to indicate you have completed the entry of a command and to execute the command.
- ⇒ Use the "up" arrow (↑) key to recall the most recent line of input to PV-WAVE. After you redisplay a line, you can edit it and then press the <Return> key to execute it. Recall up to 400 lines by pressing this key repeatedly.
- ⇒ Commas (,) are used to separate one argument from another in a PV-WAVE command.
- ⇒ Use the ampersand (&) to enter more than one command on a line.
- ⇒ To continue a command from one line to another, append a dollar sign (\$) at the end of the line you wish to continue.
- ⇒ After an error occurs in a PV-WAVE program file, program control stops at the line in the source file where the error occurred unless it is told differently. Entering the RETALL command returns program control to the main program level. Alternatively, you may enter the RETURN command, which will return you to the next highest program level. In order for a program module to be compiled, control must be at the main level in which case the RETALL command becomes very useful.
- ⇒ A semicolon (;) is used to begin a comment in PV-WAVE. Any text between the semicolon and the end of the line will be considered a comment.

Variables and Basic Data Types:

Variables are used in PV-WAVE to store your data. The typing and binding of variables in PV-WAVE is dynamic, that is, the structure and type of data contained in a variable may change during a session. The basic data types that PV-WAVE variables may have are:

Byte	An 8-bit, unsigned integer ranging from 0 to 255. Variables containing images are commonly represented as byte data.
Integer	A 16-bit, signed integer ranging from -32,768 to +32,767.
Longword	A 32-bit, signed integer ranging in value from approx. - 2 billion to + 2 billion.
Floating	A 32-bit floating-point number with approximately 6-7 decimal places of significance.
Double	A 64-bit double precision, floating point number with approximately 13-14 decimal places of significance.
Complex	A real-imaginary pair using single-precision floating point numbers. Complex numbers are useful for signal processing and frequency domain filtering.
Double Complex	A real-imaginary pair using double precision floating point numbers.
String	A sequence of alphanumeric characters, from 0 to 32,767 characters in length.

Scalars, Arrays, and Structures:

A scalar is a single instance of one of the seven data types or a single composite structure. An array is a simple structure containing multiple elements of the same data type. Array elements are addressed with subscripts and subscript ranges starting with index 0. A structure can be a combination of the basic data types, other structures, and arrays.

PV-WAVE is array-oriented, that is, it can handle an array as a single entity, rather than as separate numbers. As a result, you can perform mathematical operations on arrays in the same manner as on individual elements (without using loops). For example, to add 1 to each element in an array, the expression would be "array=array+1".

Beginning PV-WAVE:

UNIX

In order to start your PV-WAVE session you must first establish an environment for PV-WAVE to run in. This is done by executing the PV-WAVE setup file. The following are examples of executing the PV-WAVE setup file for different shell environments:

C Shell	<code>source /usr/local/vni/wave/bin/wvsetup</code>
Korn or Born Shell	<code>./usr/local/vni/wave/bin/wvsetup.sh</code>

The path /usr/local/vni is known as the VNI installation directory. The directory /usr/local is the suggested location for your VNI software installation directory but you may choose to install it in a different directory.

Once the environment for your PV-WAVE session has been established, start PV-WAVE in one of the following ways:

<code>os_prompt> wave</code>	This is if you are licensed for PV-WAVE Foundation
<code>os_prompt> waveadv</code>	This is if you are licensed for PV-WAVE <i>Advantage</i>

What will appear next is the PV-WAVE command prompt "WAVE>". PV-WAVE is now ready for you to enter commands. The command to end your PV-WAVE session is EXIT.

WINDOWS:

If PV-WAVE isn't already installed on your system, install it first. Follow the directions provided with the media you received from Visual Numerics, Inc.

On Windows NT, click the PV-WAVE Home Window icon in the PV-WAVE Program Group.

On Windows 95, 98, or 2000, use the Start button. Select Start=>Programs=>PV-WAVE 7.0=>PV-WAVE Home Window.

What will appear next is the PV-WAVE command prompt "WAVE>". PV-WAVE is now ready for you to enter commands. The command to end your PV-WAVE session is EXIT.

Understanding PV-WAVE Commands:

Valid PV-WAVE commands are similar to the types of commands in C and FORTRAN in that they can be represented as procedure/function calls or expressions. There are other command types in PV-WAVE that are valid but not like C or FORTRAN and those will be described in a later section.

A procedure call in PV-WAVE uses the format:

procedure_name, parameter1, parameter2, ..., parameterN

A function call in PV-WAVE always returns a value so the function call itself is assigned to a variable. The format of a function call is as follows

variable = function_name(parameter1, parameter2, ..., parameterN)

An expression in PV-WAVE is any standard assignment using variables, constants and an operator(s). The following is the format of an expression in PV-WAVE:

result = variable + 100

Function calls can also be used as an operand in an expression. The following illustrates this:

result = (function_name(parameter1, parameter2, ..., parameterN) + 100) / variable1

Notice that parentheses are used to order the way the expression is evaluated.

Some Example Commands to Try:

The following is an example of a PV-WAVE function call that will create a 100-element array of floating point numbers that increase by one for each element in the array:

WAVE> variable = FINDGEN(100)

The INFO procedure in PV-WAVE gives you information about a variable that exists in your PV-WAVE session. The following command will give us the type and size of the variable created:

WAVE> INFO, variable

```
VARIABLE    FLOAT    = Array(100)
```

Another method of checking the values in a variable is to print them. The PRINT procedure in PV-WAVE will display the values contained in the variable:

```
WAVE> PRINT, variable
```

The following expression, which is a combination of function calls, variables, constants and operators, results in a sin wave:

```
WAVE> sin_variable = SIN(FINDGEN(100)/16.0)
```

To see what the variable looks like as a 2-D plot, enter the following PV-WAVE PLOT command.

```
WAVE> PLOT, sin_variable
```

Other PV-WAVE command types:

There are other commands in PV-WAVE known as compiler directives. These types of commands allow PV-WAVE to do much more than analyze or display data. The following is a list of special characters that are used as compiler directives:

- . - When used as the first character after the PV-WAVE prompt, indicates that an executive command is going to be used. Examples of this are .RUN, .SIZE, .CON . These will be described in detail in a later section of this document.
- @ - Indicates that what follows the "@" is a command file that should be executed in "batch mode". A command file contains one or more single statement PV-WAVE commands. The syntax for this compiler directive is "@filename" where "filename" is the name of a file that contains PV-WAVE commands. All of the commands in the command file will be executed one by one.
- \$ - Indicates that what follows the "\$" is an operating system command. The syntax for this compiler directive is "\$os_command" where "os_command" is any valid operating system command. For example "\$ls" will give you a directory listing of the files contained in the current directory on a UNIX workstation.

Parameters used when calling a procedure or function:

In the previous example, the variable "sin_variable" was passed to the PLOT procedure to create a 2-D plot. The parameter "sin_variable" is considered a **positional parameter**. Typically, positional parameters are used to pass data to a procedure or function. Positional parameters can be variables, constants, expressions or the result of another function call. The following is an example of using positional parameters in a function call:

```
WAVE> result = REBIN(sin_variable, 50)
```

The REBIN function in PV-WAVE is used to resize a variable. In this example the variable "sin_variable" will be resized to a variable with 50 elements. The result is stored in the variable "result". Here is another more complex example:

```
WAVE> result = REBIN(sin_variable # sin_variable, N_ELEMENTS(sin_variable)/2, $  
25)
```

Notice that the "\$" is used to continue the command on the next line. Looking at each positional parameter separately gives us the following:

- | | |
|------------------------------|---|
| sin_variable # sin_variable | - The "#" operator is for matrix multiplication meaning the result of the expression "sin_variable # sin_variable" will be a two dimensional array (i.e.100x100). |
| N_ELEMENTS(sin_variable) / 2 | - The function N_ELEMENTS returns the number of elements in the variable "sin_variable". This number is then divided by 2 (i.e. 100/2 will be 50). This is used as the size of the 1st dimension of the resulting two-dimensional variable. |
| 25 | - This constant is used to specify the size of the 2nd dimension in the resulting two-dimensional variable. |

In simple terms the command would be "result = CONGRID(sin_variable # sin_variable, 50, 25). The result of this PV-WAVE function call is stored in the variable "result" which will be 50x25.

Controlling the characteristics of a procedure or function with keyword parameters:

Often it is desirable to change the way a PV-WAVE procedure or function operates. For example, a user may want to add a title to a plot or change the color of the plot. Another example would be controlling the interpolation method used in a PV-WAVE command for resizing a variable. This is accomplished with the use of **keyword parameters**. Keyword parameters are used in addition to positional parameters in a PV-WAVE command. The following example demonstrates the use of both positional parameters and keyword parameters in the SURFACE command:

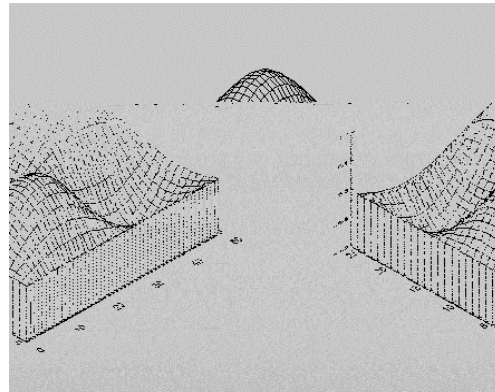
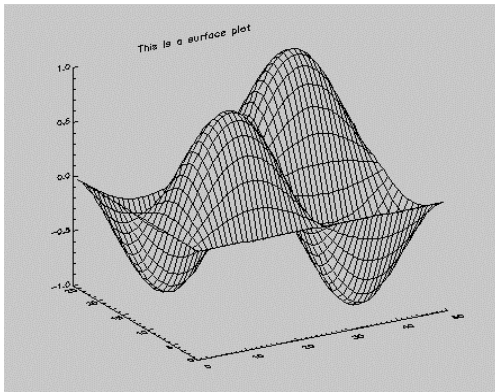
```
WAVE> LOADCT, 4  
WAVE> SURFACE, result, Title="This is a surface plot", $  
Background=WoColorConvert(236), Color=WoColorConvert(0)
```

Notice that the keyword parameter TITLE is assigned the string "This is a surface plot". The keyword parameters BACKGROUND and COLOR control the background and drawing colors and are assigned the values 236 and 0. All keyword parameters are set equal to a value when passed as an argument to a PV-WAVE command. This is different from the way that positional parameters are used; they are not set equal to a value in the calling of a procedure or function. Keyword parameters can be assigned values that are represented the same way as positional

parameters. That is, they can be specified as variables, constants, expressions or the result of another function call. Here is another example:

```
WAVE> SURFACE, result, Ax=60, Az=45, Skirt=-1, $  
Background=WoColorConvert(236), Color=WoColorConvert(0)
```

The keyword parameters Ax and Az allow you to control the rotation of the surface plot. The Skirt keyword is used to draw lines from the outside boundary of the surface to the skirt value. The following graphics show the result of both of the example surface commands:



Keyword parameters can also be “flagged”. This means that the name of the keyword parameter is preceded by a “/”. This is the same as specifying the keyword parameter be assigned the value 1. For example the following statements have the same result:

```
WAVE> result = REBIN(sin_variable, 50, Sample=1)  
WAVE> result = REBIN(sin_variable, 50, /Sample)
```

Keyword parameters that are “flagged” are the types of keyword parameters that can have only the values 0 or 1. In the previous example, the Sample keyword indicates that the REBIN function should use a nearest neighbor method when interpolating the data rather than the default interpolation method bilinear interpolation.

Some PV-WAVE commands have many keywords while other PV-WAVE commands have none. The PLOT command in PV-WAVE has 73 keyword parameters that allow you to control all aspects of how the plot is drawn. If a keyword parameter is not specified in the call to a PV-WAVE procedure, then a default value is used. For graphical commands in PV-WAVE such as PLOT, CONTOUR, SURFACE and TV, these default values are stored in system variables and can be changed.

System Variables:

All system variables in PV-WAVE begin with the “!” character and are pre-defined for you when you start your PV-WAVE session. System variables in PV-WAVE have two primary uses. The first is to maintain the default attributes of a PV-WAVE session. The second use is to store system information such as display device configuration and axis information.

System variable usage #1 -- Maintaining the default values for a PV-WAVE session

!P is a PV-WAVE system variable that contains the default plotting attributes for a graphic. This system variable is represented as a structure variable and contains the following fields:

```
WAVE> INFO, !P, /Structure
```

```

** Structure !PLT, 23 tags, 224 length:
BACKGROUNDLONG      0
CHARSIZE             FLOAT    0.000000
CHARTHICK            FLOAT    0.000000
CLIP                 LONG     Array(6)
COLOR                LONG     235
FONT                 LONG     -1
LINESTYLE            LONG     0
MULT                 LONG     Array(5)
NOCLIP              LONG     0
NOERASE             LONG     0
NSUM                 LONG     0
POSITION            FLOAT    Array(4)
PSYM                LONG     0
REGION              FLOAT    Array(4)
SUBTITLE            STRING    ""
T                   FLOAT    Array(4, 4)
T3D                 LONG     0
THICK               FLOAT    0.000000
TITLE               STRING    ""
TICKLEN            FLOAT    0.0200000
CHANNEL             LONG     0
GRIDSTYLE           LONG     0
TICKFORMAT          STRING    ""

```

Most all of the fields contained in the !P structure variable have a corresponding keyword in any of the PV-WAVE graphics commands. When a PV-WAVE graphics command is used without one of its keyword parameters, the corresponding value in the !P system variable is used. When the keyword parameter is used in the command, it overrides the default value stored in !P. The following is an example that illustrates this:

```

WAVE> !P.TITLE = "This is the default"
WAVE> PLOT, sin_variable

```

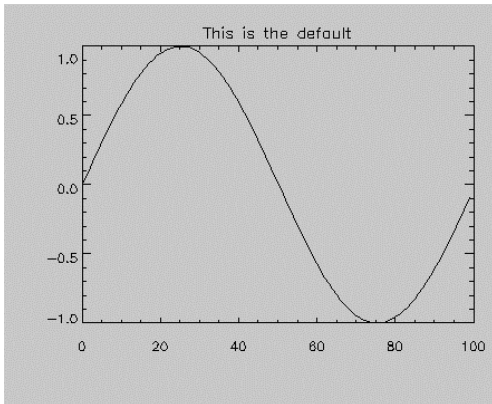
Since the Title keyword was not used, the resulting graphic uses the default value for the title stored in the system variable !P.TITLE. Using the keyword parameter Title in the PLOT command specifies that the default should not be used.

```

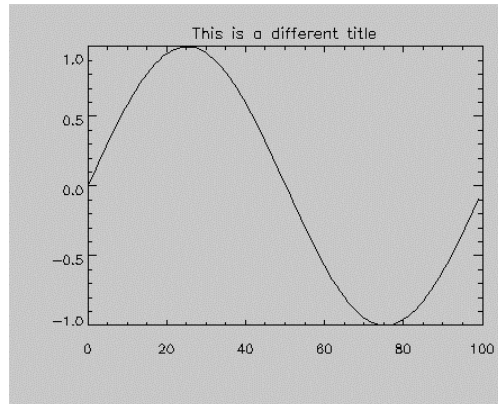
WAVE> PLOT, sin_variable, Title="This is a different title"

```

The following is the results of each of the PLOT commands:



Using the default title stored in !P.TITLE



Using the TITLE keyword

In the instance where a keyword parameter does not have a corresponding value stored in a system variable, the command itself assigns the default value.

System Variable usage #2 -- Storing system information

In some cases, PV-WAVE will store information about a graphical representation or output device in a system variable. For example, when a plot is created, PV-WAVE will automatically calculate what the range of the X, Y and Z-axis are unless you specify differently. In this case, the axis range values will be stored in the system variables !X, !Y and !Z. This may be information that could be applied to a following command for a special type of graphical representation. Another example is the system variable !D. This system variable contains information about your current interactive device. For an X display, this would be information about window sizes, colors used from the system color map and other similar information. Occasionally, this information may be something useful for a PV-WAVE program. Here is an example of a PV-WAVE system variable used to store the PV-WAVE prompt.

```
WAVE> INFO, !PROMPT
<Expression>  STRING  = 'WAVE> '

WAVE> !PROMPT = "My Prompt> "
My Prompt> INFO, !PROMPT
<Expression>  STRING  = 'My Prompt> '
```

By changing the value of the system variable !PROMPT, the PV-WAVE prompt will change. For a complete list of all of the system variables in PV-WAVE, use the command "INFO, /System" at the PV-WAVE prompt. Also, there is a complete listing of all of the system variables and their uses at the end of this document in the "Quick Reference" section.

Input and Output of data in PV-WAVE:

PV-WAVE was designed to accommodate any type of data set a user may have. There are two classes of I/O routines in PV-WAVE. The first class of I/O commands is the "DC" or data connect routines. The second class of I/O commands are the "LUN" or Logical Unit Number routines that are lower level read/write routines. The class of I/O routine to use is dependent on the organization of the data file that is being read in. The following is a list of each class and the routines supplied in them.

The Data Connect "DC" Routines:

The Data Connect routines provide a method for reading in "well behaved" data sets with a single command. There is no need to open or close the file, this is done automatically with the "DC" routines. There are many keywords that are used with the following list of "DC" commands that allow you to control the way that the data file is read in.

<u>Routine</u>	<u>Description</u>
DC_WRITE_FREE,	Write/read ASCII data to/from a file without having to choose a
DC_READ_FREE	LUN. Free format.
DC_WRITE_FIXED,	Write/read ASCII data to/from a file without having to choose a
DC_READ_FIXED	LUN. Fixed format.
DC_WRITE_8_BIT,	Write/read binary 8-bit images to/from a file with- out having to
DC_READ_8_BIT	explicitly choose a LUN.
DC_WRITE_24_BIT,	Write/read binary 24-bit data to/from a file without
DC_READ_24_BIT	having to explicitly choose a LUN.
DC_WRITE_TIFF,	Write/read TIFF image data. No LUN required.
DC_READ_TIFF	
DC_ERROR_MSG	Returns text string associated with negative status code
	generated by a "DC" data import/export function that does not
	complete successfully. R1-157
DC_OPTIONS	Sets the error message reporting level for all "DC" import/export
	functions.

The following are examples of reading in data with the "DC" routines.

EXAMPLE #1

sample data file "data1.dat":
0.000 1.000 2.000 3.000
1.000 2.000 4.000 6.000
2.000 3.000 6.000 9.000
3.000 4.000 8.000 12.000
4.000 5.000 10.000 15.000

```
WAVE> status = DC_READ_FREE("data1.dat", a, b, c, d, /Column)
WAVE> PRINT, a
    0.000000    1.000000    2.000000    3.000000    4.000000
WAVE> PRINT, b
    1.000000    2.000000    3.000000    4.000000    5.000000
WAVE> PRINT, c
    2.000000    4.000000    6.000000    8.000000    10.000000
WAVE> PRINT, d
    3.000000    6.000000    9.000000    12.000000    15.000000
```

EXAMPLE #2

sample data file "data2.dat":
**This is a header that occupies
two lines.**

**0.000 1.000 2.000 3.000
1.000 2.000 4.000 6.000**

This is some junk

This is more junk

2.000 3.000 6.000 9.000

**3.000 4.000 8.000 12.000
4.000 5.000 10.000 15.000**

```
WAVE> status=DC_READ_FREE("data2.dat",v1,v2,/Column,Get_Columns=[2,3],$  
                           ignore=["$BLANKS","$TEXT_IN_NUMERIC"])
```

```
WAVE> PRINT, v1  
    1.00000    2.00000    3.00000    4.00000    5.00000  
WAVE> PRINT, v2  
    2.00000    4.00000    6.00000    8.00000    10.00000
```

The data file contains four columns of data as well as several lines of randomly placed text and blank lines. The objective is to read columns two and four of the data file into the variables "v1" and "v2". The Get_Columns keyword allows a user to specify which columns of the data file will be read into the variables "v1" and "v2". Since the records containing text and blank lines have no value in this example, the Ignore keyword is used to specify that those records be ignored or eliminated from the resulting variables "v1" and "v2". The values \$BLANKS (for records that are blank lines) and \$TEXT_IN_NUMERIC (for records with alphabetic characters) are special values for the Ignore keyword. The Ignore keyword also accepts a user specified string providing a mechanism for skipping records in the data file containing that string. There are many more keywords that apply to this command, please refer to the Quick Reference section at the end of this document.

EXAMPLE #3

```
WAVE> elevation = FLTARR(60,40)  
WAVE> status = DC_READ_FREE("$WAVE_DATA\pikeselev.dat", elevation, /Row)  
WAVE> SURFACE, elevation
```

The data file "pikeselev.dat" in the PV-WAVE data directory is a 60x40 array of row organized floating point values. In this case, the variable "elevation" is defined ahead of time in order to indicate the type and amount of data to be read in. The keyword /Row indicates that the data file is in a row organization.

EXAMPLE #4

```
status=DC_READ_8_BIT("/usr/local/vni/wave/data/mandril.img", monkey)  
TVSCL, monkey
```

TVSCL is a command in PV-WAVE for displaying images.

The LUN or “Low Level” Routines:

In some cases, the “DC” routines will not be able to read in a data file because the organization of the data file is not “well behaved”. The data file may consist of a mix of binary and ASCII data in varying formats or varying lengths. The LUN or Logical Unit Number routines are lower level routines that allow the user to specify exactly what formats and amounts of data is read or written out. The LUN routines require that the user both open/close the data file that is being read/written. When a data file is opened, it is associated to a LUN. Further access to the file is achieved by referring to the LUN assigned to the file. This is like a file pointer in C. The following is a list of Logical Unit Numbers and their uses as well as a list of routines for opening/closing and reading/writing from data files using LUN's.

Logical Unit Numbers:

<u>LUN</u>	<u>Purpose</u>
0	Standard input stream, usually the keyboard.
-1	Standard output stream, usually the screen.
-2	Standard error stream, usually the screen.
1-99	General use in programs, or interactively.
100-128	Reserved, managed by GET_LUN and FREE_LUN procedures.

<u>Routine</u>	<u>Description</u>
OPENR	Open an existing file for input (reading) only.
OPENW	Opens a new file for input and output. In Unix, if the named file already exists, the previous contents are destroyed.
OPENU	Opens an existing file for input and output.
CLOSE	Closes a file that is opened in PV-WAVE
PRINT	Write ASCII data to the screen.
READ	Read ASCII data from the keyboard.
PRINTF	Write ASCII data to/from a specified LUN.
READF	Read ASCII data to/from a specified LUN.
READU	Read binary data from specified file unit.
WRITEU	Write binary data to specified file unit.
ASSOC	Map an array structure to a file providing direct access to binary data.
GET_KBRD	Read single characters from the keyboard.

Once a file has been opened with one of the open commands, data can be read or written to the file. The syntax of the LUN read/write routines has the following format:

```
read_command/write_command, lun, variable1, variable2, ..., variableN
```

The important thing to remember when using the LUN routines for reading data is that they will read as much data from a file necessary to satisfy the total amount of data specified in the variables list. The following is an example of reading the elevation data file from the previous example. Notice that the data file must be opened and closed and that access to the data file is through the LUN.

```
WAVE> elevation = FLTARR(60,40)
WAVE> OPENR, 1, "$WAVE_DATA\pikeselev.dat"
WAVE> READF, 1, elevation
WAVE> CLOSE, 1
WAVE> SURFACE, elevation
```

General concepts to understand before creating graphics:

Before introducing the PV-WAVE graphical commands, it is important to understand some of the basics involved with creating graphics. These basics include output devices, coordinate systems and Color.

Output Devices:

PV-WAVE provides the ability to create graphics on both the workstation display as well as a number of other interactive and hard copy devices. In order to specify which device PV-WAVE will output its graphics to, use the SET_PLOT command as follows:

```
WAVE> SET_PLOT, "device_name"
```

The string parameter "device_name" is the name of the graphics device that output will be directed to. The following is a list of device names and their descriptions

<u>Device Name</u>	<u>Description</u>
CGM	Computer Graphics Metafile format
HP	HPGL device
PCL	PCL device
PM	Pixel Map
PS	PostScript device
REGIS	REGIS terminal
TEK	Tektronix terminal
WIN32	Windows NT
WMF	Windows Metafile
X	X Window System
Z	Z-buffer pseudo device

The device "X" for the X windows system is the default graphics device when a PV-WAVE session is first started on UNIX. On Windows, the default graphics device is WIN32. Each graphics device has its own set of characteristics. For example, The X device has windows and a cursor. The PostScript device has the characteristics of a filename and orientation (landscape or portrait). In order to control these characteristics for all devices, use the DEVICE command. Appendix A of the *PV-WAVE User's Guide* has a complete listing of each device and the keywords that they support. The following is an example of creating a PostScript file of a PV-WAVE graphic:

```
WAVE> x = FINDGEN(100) & y = SIN(x/16.0)
WAVE> SET_PLOT, "PS"
WAVE> DEVICE, Filename="example.ps", /Color
WAVE> PLOT, x, y, Color=WoColorConvert(4), TITLE="This is an example for PS"
WAVE> DEVICE, /Close
WAVE> SET_PLOT, "win32"
```

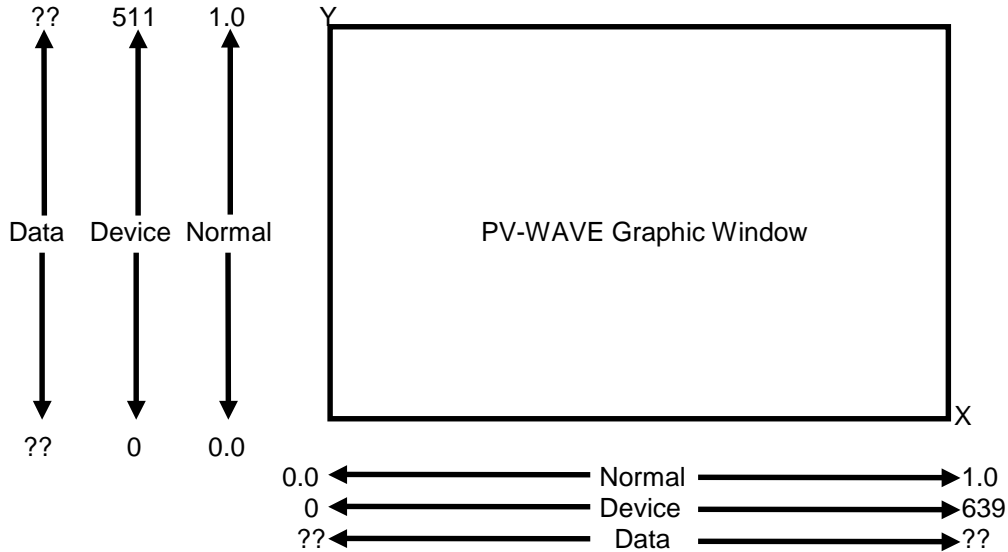
NOTE: if you are running on UNIX: SET_PLOT, 'X'

```
WAVE> $lpr example.ps
```

The DEVICE command specifies the filename and that color is enabled. When creating hard copy output, always remember to close the output file with the "DEVICE, /Close" command. If this is not included, the resulting output will be a blank page. Notice that the "\$" special character is used to spawn a command to the operating system in order to send the PostScript file to the printer.

Coordinate Systems:

PV-WAVE maintains three different coordinate systems. They are data, device, and normalized coordinates. These coordinate systems are active in the entire area of the graphics window. The following illustration demonstrates the coordinate systems and their values for a default PV-WAVE graphics window.



Values for the normalized coordinate system always range between 0 and 1 in both X and Y regardless of the size of the graphics window. Values for the device coordinate system is based on the size of the graphics window. The values are represented in device units. In the case of the X windows system, this would be pixels. The graphic above is of a default sized PV-WAVE graphics window, 640 pixels in the X direction and 512 pixels in the Y direction. The device coordinate system then ranges between 0 and 639 in the X direction and between 0 and 511 in the Y direction. The data coordinate system is defined by the data that is currently drawn in the graphics window. This will vary from one graphical representation to the next based on the values in the data set. There are only five primary graphical commands in PV-WAVE that establish a data coordinate system. They are PLOT, CONTOUR, SURFACE, SHADE_SURF and AXIS with the /Save keyword. All of the other PV-WAVE graphical commands draw graphics on an existing data coordinate system.

Many PV-WAVE graphical commands allow you to specify a location of a graphical attribute in any of the three coordinate systems. For example, the PV-WAVE command XYOUTS is used to place text in on currently existing graphic. The first two parameters of the XYOUTS command are designated as the X and Y location where the text will be placed. These values may be specified in data, device, or normalized coordinates. The following example places a text string in the center of the graphics window using normalized coordinates:

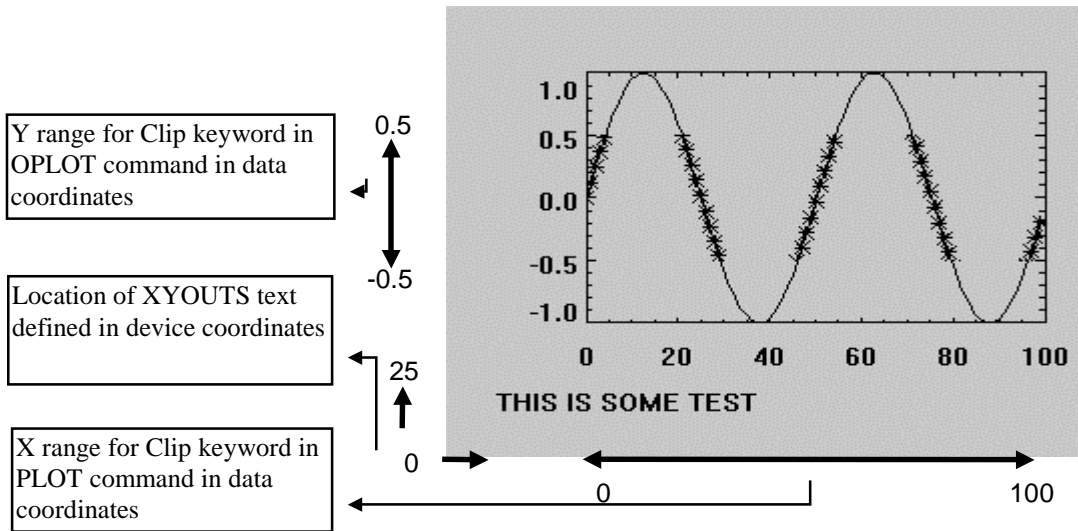
```
WAVE> XYOUTS, .5, .5, "Text String", /Normal
```

Notice that the Normal keyword is used to specify which coordinate system the X and Y values correspond to.

Another example of using a keyword to specify a coordinate system is as follows:

```
WAVE> x = FINDGEN(100)
WAVE> y = SIN(x/16.0)
WAVE> PLOT, x, y
WAVE> OPLOT, x, y, Psym=2, Clip=[0.0, -0.5, 100.0, 0.5], /Data
WAVE> XYOUTS, 25, 25, "THIS IS SOME TEXT", /Device
```

The first two commands define values for the variables "x" and "y". The OPLOT command in PV-WAVE over plots data on an existing data coordinate system. The Psym keyword in the OPLOT command is used to draw plot symbols at the data points rather than connecting them with lines. The Clip keyword defines a clipping region in data coordinates as indicated by the Data keyword. The syntax for this keyword is [x1,y1,x2,y2] where the x1-y1 pair defines the X and Y location of the lower left corner of the clip region and the x2-y2 pair defines the X and Y location of the upper right corner. Values that lie outside the clipping region are not drawn. The XYOUTS command draws a text string using device coordinates to define the location. The following graphic is the result of the example:



Color:

The PV-WAVE color system is based on the RGB model. Each color is composed of a red, green and blue component. When working with an eight-bit model, each red, green and blue component can have a value between 0 and 255. The resulting color is a combination of red, green and blue values of differing intensities. The following table illustrates how colors are created with the RGB model:

Red	Green	Blue	Color
0	0	255	Blue
0	255	0	Green
255	0	0	Red
0	255	255	Cyan
255	0	255	Magenta
0	255	255	Yellow

A color table in PV-WAVE contains many colors or RGB triplets. On Windows, all 255 colors are available. On UNIX platforms, the number of colors available is determined when the first window is opened in a PV-WAVE session. At that time, PV-WAVE queries the system color map and allocates all colors that are not being used by other applications currently running. If the number were 230, then the PV-WAVE color table for that session would have 230 colors.

The colors in the color table can be defined in many ways. The WgCeditTool command is the easiest way since it is a graphical interface tool for loading and editing the color table. The PV-WAVE command LOADCT loads one of the 16 predefined color tables and TEK_COLOR is used to define the first 32 colors of the color map.

WAVE> WgCeditTool ; example for calling the color-editing tool

24-bit Color:

Because PV-WAVE is an 8-bit colortable-based software package (instead of a “true” 24-bit software package), it performs conversions through a color lookup table internally when drawing to a 24-bit display.

PV-WAVE graphical commands:

There are many commands in PV-WAVE for creating graphical representations of a data set. Using the primary PV-WAVE graphical commands creates simple graphical representations. Using keyword parameters that control characteristics of the graphical representation creates more complex graphical representations. Also, combining the primary graphical commands in a single graphic provides even more capabilities. The following are lists of the primary PV-WAVE graphical commands:

Graphics/Plotting Routines (Two-Dimensional):

<u>Routine</u>	<u>Description</u>
AXIS	Draw an annotated axis.
O PLOT	Overplot data over existing axis.
PLOT	Plot array on new axis, set scaling.
PLOT_IO	Plot with linear-log scaling.
PLOT_OI	Plot with log-linear scaling.
PLOT_OO	Plot with log-log scaling.
VEL	Plot velocity vector field (random field).
VELOVECT	Plot velocity vector field (regular field).

Graphics/Plotting Routines (Three-Dimensional):

<u>Routine</u>	<u>Description</u>
CONTOUR	Draws contour plots.
CONTOUR2	Draws contour plots for scattered data.
IMAGE_CONT	Overlays contours on an image.
POLYSHADE	Draws shaded surface of connected 3D polygons.
RENDER	Creates a ray-traced scene.
SHADE_SURF	Draws shaded 3D surface plots.
SHOW3	Displays 2D array as a combination contour, surface, and image plot.
SURFACE	Draws 3D mesh surface plots.

Image (Raster) Display Routines:

<u>Routine</u>	<u>Description</u>
MOVIE	Animation of a 3D array "stack" of images.
TV	Raw raster image display, no scaling.
TVRD	Read image or sub-image from raster display device.
TVSCL	Raster image display. First byte scales images 0-255.
TVCRS	Manipulates the image device cursor.
TVLCT	Loads a user-defined color table into the display device.
LOADCT	Loads a predefined color table into the display device.

Other Graphics Routines:

<u>Routine</u>	<u>Description</u>
CURSOR	Read graphics cursor (mouse).
ERASE	Erase the screen or current window.
LEGEND	Adds a legend to an existing graphic.
PLOTS	Plot only 2D/3D lines, points, and/or markers.
POLYFILL	Fill defined polygons with color or pattern.
TVCRS	Set cursor position and visibility.
USERSYM	Define a symbol for plotting.
WINDOW	Create a PV-WAVE drawing window.
XYOUTS	Display graphics text/annotation at a specified location.

Example PV-WAVE graphics:

EXAMPLE #1:

This is an example that you can type in at the PV-WAVE prompt. It introduces the SURFACE, CONTOUR and SHADE_SURF command as well as the use of the Z buffer device

The following four commands read the elevation data from the "pikeselev.dat" data file into a variable called "peak"

```
WAVE> peak = FLTARR(60,40)
WAVE> OPENR, 1, "$WAVE_DATA/pikeselev.dat"
WAVE> READF, 1, peak & CLOSE,1
```

A two-dimensional data set like "peak" can be displayed in a variety of ways. First let us try looking at this data set as a meshed surface.

```
WAVE> SURFACE, peak, /Save
```

The Save keyword in the SURFACE command is used to store the 4x4 viewing transformation (viewing angle) into the system variable !P.T. Later in this example, we will want to use the 4x4 matrix to project a contour view in three-dimensions.

```
WAVE> CONTOUR, peak
WAVE> CONTOUR, peak, NLevels=20
WAVE> CONTOUR, peak, NLevels=20, /T3D
```

The NLevels keyword controls the number of contour levels that are drawn. The T3D keyword in this command informs the CONTOUR command that the view of the contour should use the viewing transformation stored in !P.T. The T3D keyword can be used with any two-dimensional graphic command such as PLOT, XYOUTS, PLOTS, and POLYFILL.

```
WAVE> SHADE_SURF, peak
WAVE> SURFACE, peak, /NoErase
```

The SHADE_SURF command displays the variable “peak” as a shaded surface.

The following SURFACE command with the NoErase keyword combines the graphical representations in one window.

```
WAVE> SHADE_SURF, peak, /Save
WAVE> CONTOUR, peak, NLevels=20, /T3D, /NoErase
```

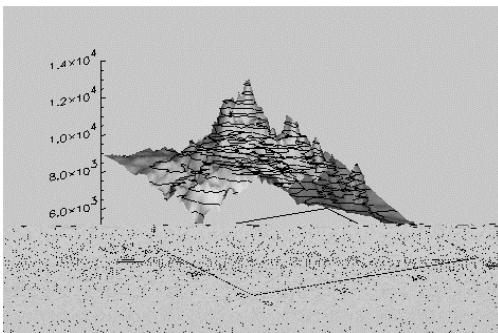
The resulting graphic combines the shaded surface representation with the three-dimensional representation of the contour. Lines from the contour that are on the back of the surface are not hidden. The Z Buffer in PV-WAVE is a method for generating graphics with hidden lines removed.

```
WAVE> SET_PLOT, 'Z'
WAVE> SHADE_SURF, peak, /Save
WAVE> CONTOUR, peak, NLevels=20, /T3D, /NoErase
WAVE> image=TVRD(0, 0, 640, 512)
WAVE> SET_PLOT, 'X'
WAVE> TV, image
```

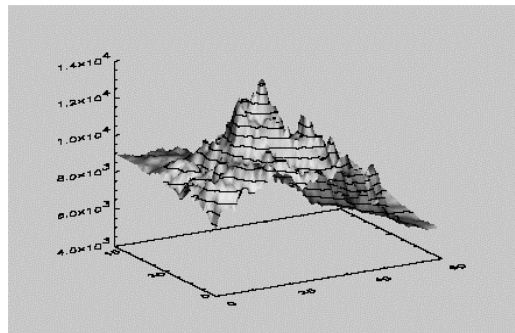
The resulting graphic now has the hidden lines removed.

The SET_PLOT command sets the current graphics device to be the Z Buffer. The Z buffer draws the graphic in a pixmap or window that is in memory (non-visible). The result of the SHADE_SURF command and the CONTOUR command will not be seen when it is drawn. In order to get the resulting image from the pixmap, the TVRD function is used.

TVRD is a command that reads the contents of a window into a two-dimensional byte variable. In this case, the variable is named “image”. The first two parameters of the TVRD command “0” and “0” represent the X and Y pixel location where the read will start. The last two parameters of the TVRD command “640” and “512” are the number of pixel columns and the number of pixel rows that should be read. The default pixmap size for the Z Buffer is 640x512 pixels. The SET_PLOT command is used again to change the interactive device back to the X windows display. The PV-WAVE command TV is used to display the image in a “visible” PV-WAVE window. The following graphics are the result of the combined SHADE_SURF and CONTOUR commands.



Commands issued to the X device



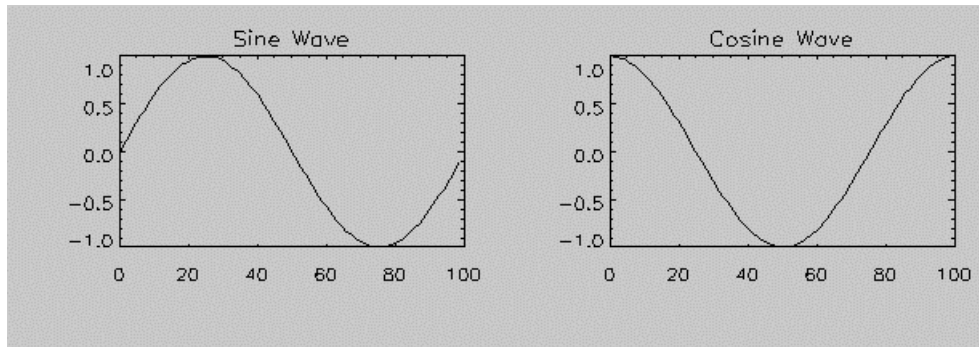
Commands issued to the Z Buffer device

EXAMPLE #2:

Sometimes it is desirable to place multiple graphical representations in the same window. This example will accomplish this.

```
WAVE> X = FINDGEN(100)/16.0
WAVE> !P.REGION = [0.0, 0.0, 0.5, 1.0]
WAVE> PLOT, SIN(x), Title="SINE WAVE"
WAVE> !P.REGION = [0.5, 0.0, 1.0, 1.0]
WAVE> PLOT, COS(x), Title="COSINE WAVE", /Noerase
```

The resulting graphic will draw the sine of the variable "x" in the left half of a PV-WAVE graphics window and the cosine of the variable "x" in the right half of a PV-WAVE graphics window. The system variable !P.REGION can be considered the graphics view port. This is the portion of the window where the graphic will be drawn. By default, the graphics view port is the entire window. By setting !P.REGION using normalized coordinates, the graphics view port can be defined to be any part of the PV-WAVE graphics window. The graphic below is the result of the commands listed above.



The format of the !P.REGION variable is as follows:

```
!P.REGION=[x1,y1,x2,y2]
```

The values "x1" and "y1" represent the lower left corner of the view port defined using normalized coordinates. The values "x2" and "y2" represent the upper right corner of the view port also in normalized coordinates. In order to set the view port back to the entire PV-WAVE graphics window, use the command "WAVE> !P.REGION=0". There is also a system variable for controlling the location of the "plot window" or location of the axis alone (!P.POSITION) as well as system variables for controlling the size of the X and Y margins (!X.MARGIN and !Y.MARGIN). The system variable !P.MULTI is an automated way to place multiple graphics in the same window. Refer to the Quick Reference section at the end of this document for more information on these system variables.

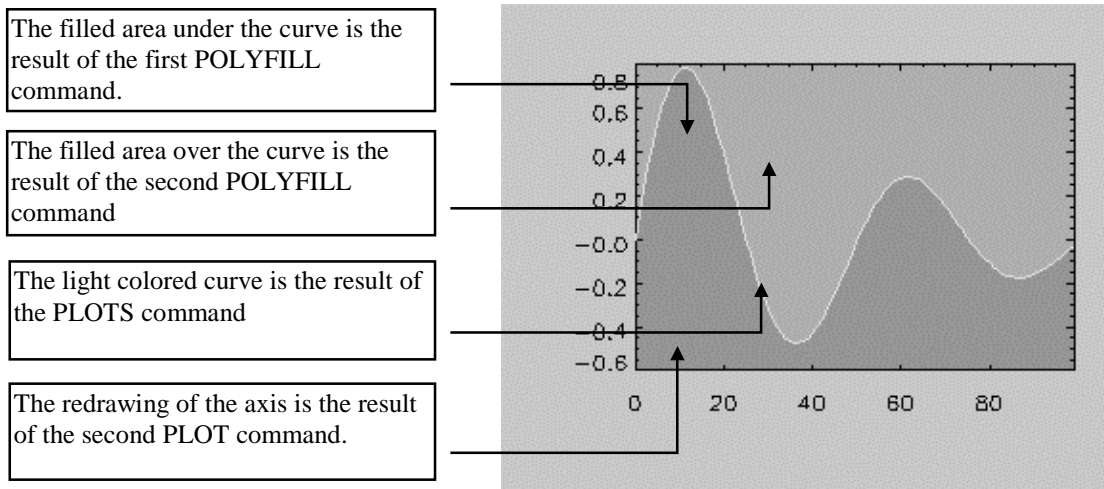
EXAMPLE #3:

As mentioned before in the discussion about coordinate systems, only a small number of the primary PV-WAVE graphical commands define a data coordinate system. Once the coordinate system has been defined, commands such as POLYFILL and PLOTS draw on the existing coordinate system. The following example demonstrates this.

```

WAVE> x = FINDGEN(100)
WAVE> y = SIN(x/8.0)/EXP(x/50.0)
WAVE> PLOT, x, y, XStyle=1
WAVE> POLYFILL, [x(0), x, x(99)], [!Y.CRANGE(0), y, !Y.CRANGE(0)], $
      Color=WoColorConvert(50)
WAVE> POLYFILL, [x(0), x, x(99)], [!Y.CRANGE(1), y, !Y.CRANGE(1)], $
      Color=WoColorConvert(100)
WAVE> PLOT, x, y, XStyle=1, /Noerase
WAVE> PLOTS, x, y, Color=WoColorConvert(150)
  
```

The resulting graphic from the combined graphical commands is shown below.



In looking a little closer at the commands that created this graphic, it is easy to determine how the graphic was created. First, the PLOT command was used with the variables “x” and “y”. This first command establishes the data coordinate system for the following commands. The next command is a POLYFILL command which fills the area between the bottom of the curve and the lower X-axis. The format of the POLYFILL command is “POLYFILL, xx, yy”, where the variables “xx” and “yy” are arrays of X and Y pairs that define the polygon to be filled. Notice that both the “xx” and “yy” parameters in the example are created using the array concatenation operators “[]”. The array concatenation operators provide an easy method for adding values to an array. In this example the filled polygons are defined by the “xx” and “yy” pairs as follows:

```

WAVE> POLYFILL, [x(0), x, x(99)], [!Y.CRANGE(0), y, !Y.CRANGE(0)], $
Color=WoColorConvert(50)
  
```

x(0) and x(99) are concatenated to the beginning and end of the array “x” defining “xx”

!Y.Crange(0) the minimum Y-axis value is concatenated to the beginning and end of the array “y” defining “yy”

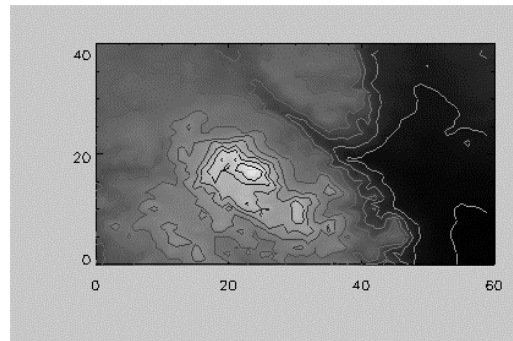
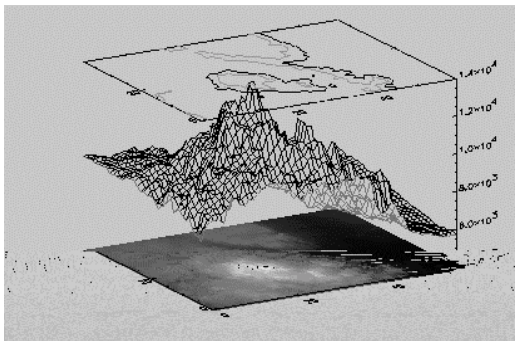
The “xx” and “yy” arrays each have 102 values defining the polygon that will be filled. The second POLYFILL command is similar except that !Y.Crange(1) is used in order to define the maximum Y-axis value. The second call to PLOT with the NoErase keyword is issued in order to redraw the axis tick marks that were covered with the POLYFILL commands. PLOTS is then used to redraw the curve itself in a different color.

EXAMPLE #4

There are a variety of commands in PV-WAVE that combine graphical commands for you. The following two graphical commands demonstrate this:

```
WAVE> peak = FLTARR(60,40)
WAVE> OPENR, 1, "$WAVE_DATA\pikeselev.dat"
WAVE> READF, 1, peak & CLOSE,1
WAVE> WINDOW, 1
WAVE> SHOW3, peak
WAVE> WINDOW, 2
WAVE> IMAGE_CONT, peak
```

The following graphical representations are the result of the SHOW3 and IMAGE_CONT commands.



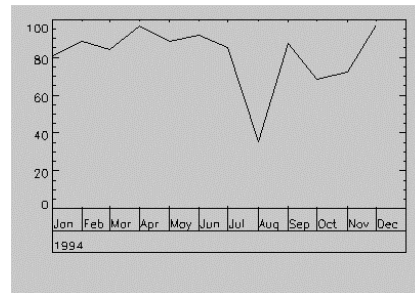
In the example above, the WINDOW command was used twice to create graphics windows. PV-WAVE will allow up to 32 windows to be open at any one time. The windows are indexed from 0 to 31. If you issue a graphics command when no window is currently open, PV-WAVE will automatically create a window for you using window ID 0. When more than one window is open in a PV-WAVE session, only one of them is considered active. This means that graphical commands will draw in only this window. To change the active window, use the WSET command with the syntax “WSET, windowid”. To delete a window in PV-WAVE use the WDELETE command with the syntax “WDELETE, windowid”. PV-WAVE also has graphical commands for creating Date-Time axis as well as creating maps.

Date-Time Plots:

Often data may be collected over a period of time. It may be desirable to plot this data as a function of time. The following is an example that plots random data as a function of time:

```
WAVE> temp=RANDOM(12) * 100
WAVE> dates=DTGEN(VAR_TO_DT(1994,1,1,0,0,0), 12, /Month)
WAVE> PLOT, dates, temp, /Box
```

The resulting graphic is shown on the right. The DTGEN command in PV-WAVE is used to create an array of 12 date time variables in increments of one month. The first parameter is the start date for the sequence of dates. The VAR_TO_DT function is used to specify the date January, 1, 1994. The values of the variable "temp" are plotted as a function of the variable "dates" automatically with the PLOT command.



Mapping Commands:

The PV-WAVE mapping procedures let you create a variety of mapping applications. Scientific data, economic data, aerial photography data and other kinds of data can be plotted with the maps generated by PV-WAVE. Two data sets are included with PV-WAVE for creating world and US maps: The World Databank II data set for global maps, and a data set based on the USGS Digital Line Graph data for U.S. maps. Additionally, there is a template for reading in your own map data set. The PV-WAVE mapping functionality supports 17 different types of projections and provides the ability for you to define your own. The following is a list of the mapping projections in PV-WAVE:

Index	Projection	Index	Projection
1	Equidistant Cylindrical	11	Oblique Azimuthal Equidistant Oblique
2	Lambert Conformal Conic	12	Polar Azimuthal Equidistant Oblique
3	Cylindrical Mercator	13	Polar Azimuthal Equal-Area
4	Sinusoidal	14	Oblique Azimuthal Equal-Area
5	Albers Equal-Area Conic	15	Transverse Mercator
6	Polyconic	16	Mollweide (Ellipsoid)
7	Polar Stereographic	99	Satellite (3D mapping onto a spheres)
8	Oblique Stereographic	-1	User-defined projection
9	Oblique Orthographic	0	No projection
10	Polar Orthographical		

The PV-WAVE mapping procedures include the following:

<u>Routine</u>	<u>Description:</u>
MAP	Plots map data with a specified projection.
MAP_CONTOUR	Plots contours on a map.
MAP_PLOTS	Plots vectors or points on the current map projection.
MAP_POLYFILL	Fills specified regions of a map.
MAP_REVERSE	Converts X-Y coordinate data to longitude and latitude coordinates.
MAP_VELOVECT	Plots a two-dimensional vector field on a map.

MAP_XYOUTS	Adds text to a map
USGS_NAMES	Queries a database of longitude/latitude coordinates for states, countries, cities and towns in the United States.

Analysis commands in PV-WAVE:

PV-WAVE's real strength is its ability to quickly and easily manipulate and analyze data. This ability includes the way in which arrays are accessed as well as PV-WAVE's extensive list of analysis commands.

Handling arrays in PV-WAVE:

PV-WAVE is an array-oriented language. What this means is that PV-WAVE provides methods for accessing values in an array that are very powerful and convenient. Information on how array values are accessed follows.

- Arrays are accessed in column major ordering (i.e. array(column, row))
- Arrays are referenced starting at subscript 0 like in C
- Whole dimensions are referenced with the "*" symbol.
- Partial dimensions (subscript ranges) are referenced with the ":" symbol.
- Multi-dimensional arrays can be referenced with a single subscript.
- Arrays can be subscripted with other arrays

Examples:

```

WAVE> var = FINDGEN(5,3) & PRINT, var
  0.00000  1.00000  2.00000  3.00000  4.00000
  5.00000  6.00000  7.00000  8.00000  9.00000
 10.0000  11.0000  12.0000  13.0000  14.0000
WAVE> PRINT, var(0,1)
  5.00000
WAVE> PRINT, var(*,1)
  5.00000  6.00000  7.00000  8.00000  9.00000
WAVE> PRINT, var(2:4,1)
  7.00000  8.00000  9.00000
WAVE> PRINT, var([0,2,4],1)
  5.00000  7.00000  9.00000
WAVE> PRINT, var(12)
 12.00000
WAVE> PRINT, var(1:6)
  1.00000  2.00000  3.00000  4.00000  5.00000  6.00000

```

There are array-processing commands in PV-WAVE for doing some basic analysis very quickly. It is recommended that these commands be used instead of using FOR loops. The following is a short list of the more frequently used analysis commands:

<u>Routine</u>	<u>Description</u>
MIN	Returns the minimum value(s) in an array
MAX	Returns the maximum value(s) in an array
TOTAL	Returns the sum of all elements in an array
AVG	Returns the average of all the values in an array
BYTSCAL	Scales value in an array to be in the range of a byte variable
MEDIAN	Returns the median value of an array
STDEV	Returns the standard deviation of an array

SQRT	Returns the square root of the value(s) in a variable
REBIN	Resizes an array using bilinear interpolation
CONGRID	Resizes an array using nearest neighbor sampling

There are two other very powerful commands in PV-WAVE for processing arrays. The functions SORT and WHERE both return an array of subscripts. For the SORT command, the array of subscripts index the values in the array in sorted order. The WHERE function returns an array of subscripts that index the values in the array that meet a specified condition.

```

WAVE> var = [1, 2, 3, 4, 3, 2, 1]
WAVE> PRINT, var
  1 2 3 4 3 2 1
WAVE> indx = WHERE(var GT 2)
WAVE> PRINT, indx
  2 3 4
WAVE> PRINT, var(indx)
  3 4 3

```

The argument to the WHERE function is a condition “var GT 2”. This condition is for the values in the variable “var” that are greater than 2. The resulting variable “indx” contains the subscripts of the values 3, 4, 3 since they are greater than 2. This variable “indx” can then be used to subscript the variable “var” to retrieve the values that are greater than 2.

Advanced Math and Statistics:

PV-WAVE command language has a variety of commands for doing mathematics and statistics. These routines are based on the *C Numerical Recipes* library. PV-WAVE *Advantage* provides a very complete very robust library of mathematical and statistical commands. This extension to PV-WAVE is the IMSL mathematical and statistical libraries. The following is a list of the types of analysis a user can perform with PV-WAVE *Advantage*:

<u>Mathematics:</u>	<u>Statistics</u>
Linear Systems	Basic Statistics
Eigensystem Analysis	Random Number Generation
Interpolation and Approximation	Correlation and Regression
Quadrature	Analysis of Variance
Differential Equations	Time Series and Forecasting
Transforms	Multivariate Analysis
Nonlinear Equations	Probability Distribution Functions and Inverses
Optimization	

The following is an example of fitting a meshed surface over random X, Y and Z data using the PV-WAVE *Advantage* function SCAT2DINTERP:

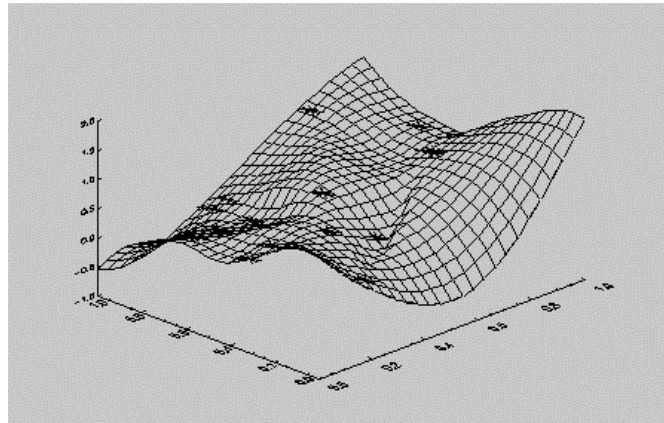
```

WAVE> x = RANDOM(15) ; Create the random data to grid
WAVE> y = RANDOM(15)
WAVE> z = RANDOM(15)
WAVE> xydata = FLTARR(2,15) ; Store the “x” and “y” variables into the
WAVE> xydata(0, *) = x ; format SCAT2DINTERP is expecting,
WAVE> xydata(1, *) = y ; a 2x15 array
WAVE> xout = FINDGEN(25)/24 ; Create the output variables “xout” and
WAVE> yout = FINDGEN(25)/24 ; “yout” for the SURFACE command
WAVE> zz = SCAT2DINTERP(xydata, z, xout, yout) ; Grid the data

WAVE> SURFACE, zz, xout, yout, /Save ; Display the resulting 2-D variable and
WAVE> PLOTS, x, y, z, /T3D, Psym=2 ; Overplot the data values for x, y and z

```

The resulting graphical output from these commands is below:



For a complete listing of the functionality in PV-WAVE *Advantage*, reference the *PV-WAVE Advantage Reference Guide* in the on-line help system or in hard copy.

Writing PV-WAVE Programs

Writing programs in PV-WAVE is very much like writing programs in any other programming language. PV-WAVE has the same type of constructs (FOR, WHILE, REPEAT, IF THEN, IF THEN ELSE, CASE etc..) that you would find in C or FORTRAN. Additionally, PV-WAVE also has the similar types of mathematical, logical and Boolean operators. The following are lists of PV-WAVE operators and programming constructs:

Operators:

Priority	Operator	Description
1	()	Parenthesis for ordering the evaluation of expressions
1	^	Caret -Exponentiation
2	*	Asterisk - Multiplication
2	#	Pound sign - Matrix multiplication
2	/	(Forward) slash - Division
2	MOD	Modulus or modulo
3	+	Plus sign - Addition
3	-	Minus sign - Subtraction, and unary minus/negation
3	<	Less-than sign - Comparative relational minimum
3	>	Greater-than sign - Comparative relational maximum
3	NOT	Boolean negation
4	EQ	Relational Equality
4	NE	Relational Not Equal
4	LE	Relational Less than or Equal
4	LT	Relational Less Than
4	GE	Relational Greater than or Equal
4	GT	Relational Greater Than
5	AND	Boolean AND (union)
5	OR	Boolean OR (intersection or inclusion)
5	XOR	Boolean exclusive OR (mutual exclusivity)

Programming Constructs:

Block Statement:

```
BEGIN
  statements
END
```

The BEGIN / END block is used for enclosing multiple statements within a construct. This enables the compiler to determine which statements will be executed in a loop or conditional construct.

IF, IF THEN and IF THEN ELSE Statements:

```
IF expression THEN statement

IF expression THEN statement ELSE statement

IF expression THEN BEGIN
  statement 1
  statement N
ENDIF ELSE BEGIN
  statement 1
  statement N
ENDELSE
```

Both the simple and compound forms of the IF construct are shown. Notice that ENDIF and ENDELSE can be used instead of using just END. Using ENDIF and ENDELSE is good programming practice since matching is performed at compile time.

CASE Statement:

```
CASE expression OF
  expression 1 : statement
  expression N : statement
  ELSE       : statement
ENDCASE
```

The CASE statement in PV-WAVE is much like a “switch” statement in C. The value of “expression” is compared against several possible matches, each with a different course of logic. When a match is found, the statement(s) associated with the match are executed. The ELSE clause is what will be executed if no match is found.

FOR statement:

```
FOR var=start, stop DO statement

FOR var=start, stop_2, step DO statement

FOR var=start, stop DO BEGIN
  statement 1
  statement N
ENDFOR
```

Three variations of the FOR statement are shown here. The variable “var” is the index variable for the loop. The value “start” represents the initial loop value, the value “stop” represents the terminating loop value and the value “step” represents the stepping value of the loop.

WHILE statement:

```
WHILE expression DO statement
```

```
WHILE expression DO BEGIN
  statement 1
  statement N
ENDWHILE
```

Both the single and compound statement formats are shown for the WHILE construct.

REPEAT statement:

```
REPEAT statement UNTIL expression
```

```
REPEAT BEGIN
  statement 1
  statement N
ENDREP UNTIL expression
```

Both the single and compound statement formats are shown for the REPEAT construct.

GOTO statement:

```
GOTO, label
```

When the GOTO statement in PV-WAVE is encountered, program control will jump to the line where the label is located.

COMMON statement:

```
COMMON block_name, var1, var2, ..., varN
```

The COMMON construct in PV-WAVE is similar to that in FORTRAN. PV-WAVE program modules that have the same common block definition in them have global access to the variables in the common block. For example, if procedure A and procedure B have the same common block definition “COMMON program_images, image1, image2”, then procedure A and procedure B will have global access to the variables “image1” and “image2”. Any other procedure or function that does not have this common block definition will not have global access to these variables.

PV-WAVE Procedures, Functions and Main Programs:

Having been introduced to PV-WAVE operators and programming constructs, it is time to learn about the mechanics of a PV-WAVE program module. The remainder of this section focuses on the details of PV-WAVE procedures, functions, and main programs.

Memory Usage:

Procedures, functions and main programs are considered program modules in PV-WAVE. Each program module has a program code area and a data area. The program code area is the memory allocated for the program module in its compiled form. The data area is the memory allocated for the program module for tracking local variables. If you get the following error message:

“Program Code Area Full”

This means that the PV-WAVE program module that you have created is too large to fit in the area that was allocated for it. There are two solutions to this problem. The first is to break the program module up into smaller pieces, each having its own allocation of program code area. The second solution is to use the executive command “.SIZE”. This will increase the total program code area to whatever you like. Similarly, if you get the error message:

“Data Area Full”

You will need to change the size of the data area with either “.SIZE”, “.LOCALS” or “..LOCALS”.

Defining Procedures, Functions and Main Programs:

A generic definition of a PV-WAVE procedure is as follows:

```
PRO procedure_name, param1, param2, ..., paramN
    statement 1
    statement 2
    .....
    statement N
END
```

Notice that a PV-WAVE procedure has a header and an END statement. These are required. In the procedure header you will notice that there is a procedure name. Additionally, there are parameters. These parameters may be either positional or keyword parameters. We will discuss the details of how positional parameters and keyword parameters are defined in a moment.

The following is a generic definition for a PV-WAVE function:

```
FUNCTION function_name, param1, param2, ..., paramN
    statement 1
    statement 2
    .....
    statement N
    RETURN, value
END
```

The definition of a PV-WAVE function is very similar to that of a procedure. They both have a header and END statement. The difference of course is that its header contains the word FUNCTION. A function always returns a value. The RETURN statement in the function is where the return value for the function is specified. The RETURN statement in a function is mandatory.

The last program module to define is the PV-WAVE main program. It has a definition as follows:

```
PRO main
    statement 1
    statement 2
    . . . . .
    statement N
END
```

The main program is used as the top level or beginning program module. It cannot be passed parameters. A unique characteristic of the main program is that it uses the same program code area and data area that are used when entering in commands at the PV-WAVE prompt. All of the variables that are created in the main program when it is executed are accessible after the completion of the main program. This program module is referred to as "\$MAIN\$".

Not every PV-WAVE program module needs to be stored in it's own file. As a matter of fact, PV-WAVE program modules should only be stored in there own files if they are to be used by other PV-WAVE applications. It's easier to develop and maintain PV-WAVE code when it is contained in a single file. The following simple example contains two PV-WAVE procedures and one function as well as a main program at the end.

```
PRO read_input, p1
    READ, "Please Enter a Single Value: ", p1
END

FUNCTION add_one, p1
    RETURN, p1 + 1
END

PRO print_sum, p1, p2
    PRINT, p1 + p2
END

PRO simple
    read_input, data1
    read_input, data2
    data3 = add_one(data1)
    data4 = add_one(data2)
    print_sum, data3, data4

END
```

If this entire example were contained in a single file named "simple.pro" it could be compiled and executed by entering the following command:

```
WAVE> .run simple
% Compiled module: read_input
% Compiled module: add_one
% Compiled module: print_sum
% Compiled module: simple
Please Enter a Single Value: 4
Please Enter a Single Value: 8
14
```

The values 4 and 8 are user input from the READ statement in the procedure "read_input". The value 14 is the result of the PRINT statement in the "print_sum" procedure.

Defining Positional and Keyword Parameters in a Procedure or Function:

The procedures and function in the previous example contained positional parameters. Positional parameters in PV-WAVE are like parameters in C or FORTRAN. The order in which they are included in the calling statement makes a difference. The same is true with PV-WAVE. Unlike C or FORTRAN, positional parameters in PV-WAVE do not have to be pre-declared to be of any particular type or size. This makes procedures or functions with positional parameters more flexible in the way they handle different types of parameters. If a parameter is to be of a specific data type, it is the responsibility of the programmer to do the appropriate error checking. The definition of positional parameters is the same for a procedure and a function.

The following is a sample PV-WAVE procedure that expects six positional parameters. The first three parameters will be X, Y and Z data to be gridded. The last three positional parameter will be the resulting two-dimensional gridded variable and the related x and y-axis variables.

```
PRO grid_data, x, y, z, gdata, xout, yout

    xydata = FLTARR(2,15)
    xydata(0, *) = x & xydata(1, *) = y
    xout = FLTARR(25)/24 & yout = FLTARR(25)/24
    gdata = SCAT2DINTERP(xydata, z, xout, yout)

END
```

The following syntax could be used to call this. It is assumed that the variables “x”, “y” and “z” are one-dimensional arrays of the same length:

```
WAVE> grid_data( x, y, z, zz, xout, yout)
```

Keyword parameters defined with the format “keyword=keyword”. This is easy to remember since keyword parameters are always assigned a value in a procedure or function call. Keyword parameters are typically used to control the way in which a procedure or function operates. The following is an example of the format used when defining keyword parameters in a procedure:

```
PRO display_3d, array, type=type, clr=clr

    CASE type OF
        1 : SURFACE, array, Color=clr
        2 : CONTOUR, array, Color=clr
    ENDCASE

END
```

The following examples could be used to call this procedure. The variable “data_2d” must be a two dimensional ver

The following is an example of checking positional and keyword parameters:

```
PRO display_3d, array, type=type, clr=clr

    sz=SIZE(array)
    IF sz(0) NE 2 THEN BEGIN PRINT, "Invalid dimension for variable" & RETURN

    IF N_ELEMENTS(type) EQ 0 THEN type = 1
    IF N_ELEMENTS(clr) EQ 0 THEN clr = 1

    CASE type OF
        1 : SURFACE, array, Color=clr
        2 : CONTOUR, array, Color=clr
    ENDCASE

END
```

Compiling and the Command Interpreter:

There are two different ways a procedure or function can be compiled in PV-WAVE. The first is automatically the second is explicitly. In order for a PV-WAVE procedure or function to compile automatically when called, the filename must match the procedure name and include the ".pro" extension. It must also be in the current working directory or in the PV-WAVE search path. For example:

```
PRO example_sum, param1, param2
    result = param1 + param2
    PRINT, result
END
```

This sample PV-WAVE procedure would be stored in the file "example_sum.pro". The filename is the same as the procedure name with a ".pro" extension. The file would be located in the current working directory.

When this procedure is called, it will automatically be compiled and executed:

```
WAVE> example_sum, 1, 2
%   Compiled module: EXAMPLE_SUM.
    3
```

PV-WAVE prints the message that the file has been compiled and then prints the output from the example procedure. The second way that a PV-WAVE procedure or function can be compiled is explicitly. This is done with the PV-WAVE executive command ".RUN". For example:

```
WAVE> .RUN example_sum
%   Compiled module: EXAMPLE_SUM.
```

When explicitly compiling with the ".RUN" command in PV-WAVE, the procedure or function is not executed, it is only compiled. To execute it, just call it (i.e. WAVE> example_sum, 1, 2). A PV-WAVE main program behaves slightly different than PV-WAVE procedures and functions when compiled with the ".RUN" command.

The “.RUN” command with a main program both compiles and executes the main program. For example:

```
PRO example_main
  a = 10
  b = INDGEN(10)
  example_sum, a, b
END
```

This PV-WAVE program would be in a file called “example_main.pro”. Notice that this main program calls the example procedure “example_sum” from the previous lesson.

In order to compile and execute this example main program, use the following syntax:

```
WAVE> .RUN example_main
%   Compiled module: EXAMPLE_MAIN
%   Compiled module: EXAMPLE_SUM.
    10 11 12 13 14 15 16 18 19
```

When developing a PV-WAVE program, the “.RUN” command will be used quite often. This is because changes will need to be made to program source code as the application is being created and tested. Before the changes can be tested, the program will need to be recompiled. Similar to other programming languages, PV-WAVE retains an “object” version of the program. This is not in the form of a file, but rather, it is in the form of a “saved” procedure or function stored in memory. The command “INFO, /Routines” will give you a listing of the PV-WAVE programs currently stored in memory.

In some instances, a run time error will occur which you will want to correct. After correcting the source file and using the “.RUN” command to recompile, you may get the error message:

“Routine may not be compiled while active”

The reason for this error is that program control stopped within the routine that had the error. Trying to recompile within the scope of this return is not possible. In order to recompile the module without getting this error message, use the RETALL command. The RETALL command will return program control to the main program level \$MAIN\$.

Each time a PV-WAVE session is started, the program modules that will be used must be compiled and stored in memory. The compilation step can be bypassed by saving the “object” version of your PV-WAVE program to file. This is done with the COMPILE command. The following is the process for accomplishing this:

```
WAVE> .RUN example_sum
WAVE> COMPILE, “example_sum”
```

The first step is to save the program module in PV-WAVE memory using the “.RUN” command. Next, the COMPILE command accepts the name of the saved procedure as a string parameter. The result will be a file with the procedure name and a “.cpr” extension (i.e. example_sum.cpr).

The following is a list of other executive commands in PV-WAVE and their uses:

<u>Command</u>	<u>Action</u>
.RNEW	Erases main program variables then executes like .RUN.
.CON	Continues execution of a stopped program.
.GO	Executes previously compiled main program from the beginning of the program.
.STEP	Executes a single statement. Can be abbreviated ".S".
.SKIP	Skips over the next statement and then single steps.
.SIZE	Resizes code area and data area used to compile programs, in terms of bytes. Usage: .SIZE code_size data_size.
.LOCALS	Resizes data area in terms of local variables and common block symbols. Usage: .LOCALS local_vars common_symbols.
..LOCALS	A compiler directive used in a program to dynamically add space for local variable and common block symbols at runtime. Like .LOCALS but not an executive command.

In developing your PV-WAVE applications, you may find it useful to use the PV-WAVE debugger. The PV-WAVE debugger is a development environment for creating, testing and maintaining PV-WAVE applications. With easy-to-use mouse and menu driven functions, the debugger helps you to become a more productive PV--WAVE application developer.

NOTE: The debugger is only available for the UNIX platform.

The following are a few features of the PV-WAVE debugger:

- Edit source files using a built-in editor or an editor of your choice.
- Copy, cut, paste, select and search for text.
- Run an application, step through it line by line, skip lines or stop execution.
- Set breakpoints and examine variables contents during program execution.
- List information about system variables, structure definitions, open files and compiles routines.
- Print source code files.

In order to start the PV-WAVE debugger, enter the command "wavedbg" at the operating system prompt.

Widget commands in PV-WAVE:

Wave widgets are building blocks that allow the PV-WAVE application programmers the ability to create a graphical user interface. They can be thought of as a user-interface object, such as a dialog box, a button box or a file selection tool. These user-interface objects can be created and arranged any way allowing unlimited possibilities in the type of interface that is created.

In greater detail, Wave widgets are a set of PV-WAVE functions that take advantage of all classes and resources supported by MOTIF. Characteristics such as color, font, position and spacing are controlled with keyword parameters. There are three levels of commands that an application programmer can use. They are as follows:

- WG commands - Provide access to high level routines. These routines are actual Wave widget applications themselves. They can be called interactively or from within another Wave widget program.
- WW commands - Provide access to the building blocks of a Wave widget application. These routines are very extensive providing a large selection of widget types. They are easy to use and require no knowledge of MOTIF.
- WT commands - Lower level access to MOTIF providing the ability to create and modify widgets not supported by the WW commands. These routines require a good knowledge of MOTIF.

The following is a list of the WG commands in PV-WAVE. Please refer to the *PV-WAVE Reference Volume 2* for more complete information on these graphical user interface applications.

<u>Routine</u>	<u>Description</u>
WGANIMATETOOL	Creates a window for animating a sequence of images
WGCBARTOOL	Creates a simple color bar that can be used to view and interactively shift a PV-WAVE color table.
WGCEDITTOOL	Creates a full-featured set of menus and widgets enclosed in a window; this allows you to edit the values in PV-WAVE color tables in many different ways.
WGCTTOOL	Creates a simple widget that can be used interactively to modify a PV-WAVE color table.
WGISOSURFTOOL	Creates a window with a built-in set of controls; these controls allow you to easily view and modify an iso-surface taken from a three-dimensional block of data.
WGMOVIETOOL	Creates a window that cycles through a sequence of images.
WGSIMAGETOOL	Creates two windows: 1) a scrolling image window and 2) an optional smaller window that shows a reduced view of the entire image.
WGLICETOOL	Creates a window with a built-in set of controls; these controls allow you to easily select and view "slices" from a three-dimensional block of data.
WGSTRIPTOOL	Creates a window that displays data in a style that simulates a real-time moving strip chart.
WGSURFACETOOL	Creates a surface window with a built-in set of controls: these controls allow you to interactively modify surface parameters and view the result of those modifications.
WGTEXTTOOL	Creates a scrolled window for viewing text from a file or character string.

These WG commands are written in PV-WAVE using WW commands and WT commands. To see the source code files for these programs, look in the “.../vni/wave/lib/std/guitools” directory. There is also a directory that contains more Wave widget interface programs that are contributed by our users. This directory is “.../vni/wave/lib/user/guitools”. The “.../vni” represents the complete path to the VNI main directory. For example, “/usr/local/vni”. As mentioned before, the WW commands in PV-WAVE are the building blocks or basics for building your own Wave widget application. The following is a complete list of the widget types that are supported in PV-WAVE:

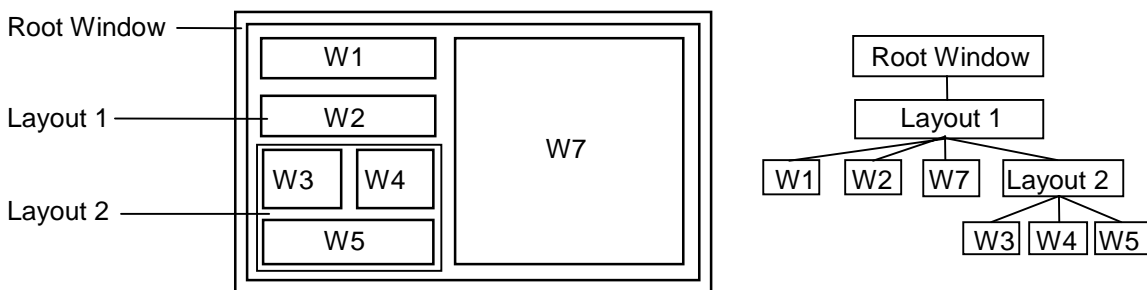
Button Box	Layout Widget	Popup Menu
Command Widget	List Box	Radio Box
Controls Box	Main Window	Table Widget
Dialog Box	Menu Bar	Text Area
Drawing Box	Message Box	Text Box
File Selection Box	Option Menu	

Each of these widget types has a PV-WAVE WW command function associated with it. Additionally, there are some control WW command functions. These routines allow control of the event loop, obtaining/assigning values of widgets and any other control mechanisms.

The WT command set is referred to as the Widget Toolbox. It is an application programmers interface (API) that is built on top of the OSF/MOTIF Graphical User Interface (GUI) toolkits. The widget toolbox consists of PV-WAVE system routines that give you access to all the widget types supported by the MOTIF toolkit. This access includes adding event handlers, adding callback routines, using timers and getting/setting widget resources not accessible with the WW commands.

The Widget Hierarchy:

The relationship between widgets in a graphical user interface is represented as a hierarchy. This is in the form of a “parent/child” relationship. At the top of the widget hierarchy is the root window. A root window is the parent of a layout widget. Layout widgets can be thought of as containers that organize the other widget types. The following graphical representation is of a generic widget interface and next to it is the hierarchical relationship of the widgets:



When a call is made to define a widget, the widget ID of its parent is defined as the first parameter in the functions argument list. The parent/child relationship is required for all Wave widget applications.

Each widget type that you create in your application must have a callback routine. This is a PV-WAVE procedure that will be executed when the widget is manipulated. For example, if a button were pressed on the interface, the buttonbox callback routine would be executed. All callback routines for Wave widgets are defined with two parameters. The first is the widget id of the widget that was manipulated. The second is a value that is based on the widget that was manipulated. For example, if a button was pressed and the buttonbox callback procedure was called, the first parameter in the callback procedure would contain the widget id of the button that was pressed and the second parameter would contain the number of the button that was pressed. If the button box had three buttons and the second button was pressed, the number of the button would be 2.

The remaining details regarding the creation of a Wave widget interface are a few commands that every widget interface program must have.

```
top = WwInit("Example","Example",layout)
```

WwInit is a routine that initialized the Wave widget functionality and makes a connection to the X server. The returned value of the WwInit function "top" is the root window id. The first and second parameters "Example" and "Example" are the application name and class name of the application. This is for use with a resource file. If you do not plan to use a resource file with your widget application, these values are arbitrary. The last parameter "layout" is the widget id of the layout or container widget. It is the child of the root window and the parent of the widget types that are part of the interface.

```
status = WwSetValue(top, /Display)  
status = WwSetValue(top, /Close)
```

These two commands accept the root window id "top" as a parameter. The Display keyword indicates to the WwSetValue command that the application should be displayed. The Close keyword indicates to the WwSetValue command that the application should be closed or exited.

```
WwLoop
```

This command enables the event loop. Events are mouse button clicks on the interface. When a widget is manipulated with the mouse, an event is registered in the event loop and the appropriate callback routine is executed.

The following is a very simple example that creates a Widget application with a two buttons. The first button will print "YOU PUSHED ME" the second button is for exiting the application.

```

PRO ButtonCB, wid, data
COMMON widget_ids, top
  CASE data OF
    1 : PRINT, "YOU PUSHED ME"
    2 : status = WwSetValue(top, /Close)
  ENDCASE
END

PRO widget_ex
COMMON widget_ids, top

  top = WwInit("Example", "Example", layout, /Vertical)
  bid = WwButtonBox(layout, ["Print", "Exit"], "ButtonCB")

  status = WwSetValue(top, /Display)

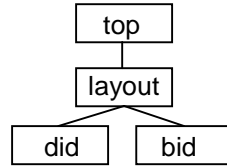
  WwLoop
END

```

Notice that in the example a COMMON block is used to make the widget id of the root window global between the main program and the buttonbox callback routine. The parameters that are used for the Ww commands like WwButtonBox vary from one widget type to the next. In the case of the WwButtonBox, the first parameter is the widget id of its parent, the second parameter is an array of strings defining the labels on the buttons. The number of labels that are specified defines the number of buttons that will be in the buttonbox. The last parameter in the WwButtonBox command is the name of the PV-WAVE callback procedure.

The parameters passed to the WwDrawing command are the widget ID of the parent "layout", the PV-WAVE window ID "1", the name of the callback procedure "DrawCB", the size of the window "[512,512]" and the size of the image that will be placed in the window "[512,512]".

The following is the widget Hierarchy of the modified example:



The widget ID "top" is the returned value of the Wwlnit command. The function Wwlnit also returns the layout widget id "layout". This widget ID is then supplied in the creation of the button box (WwButtonBox) returning the widget id "bid" and in the creation of the graphics window (WwDrawing) returning the widget id "did". It is important to understand the "parent/child" relationship between widgets and how the widget ID's are obtained.

With the Wg commands, Ww commands and the Wt commands, it is possible to create just about any graphical user interface desired. More advanced WAVE widget applications have a combination of the different widget commands. The PV-WAVE demonstration system "gallery" is a perfect example of this. The entire interface is written using PV-WAVE widgets and the source code is available for your use. It is located in the directory ".../vni/wave/demo/gallery3".

Interfacing PV-WAVE with External Programs:

PV-WAVE provides the capability of calling external C or FORTRAN programs as well as having C or FORTRAN programs calling PV-WAVE.

The following table summarizes the methods of interapplication communication that can be used with UNIX or OpenVMS.

Method	UNIX	OpenVMS	Use
SPAWN	Yes	Yes	A system routine that executes an external program from within PV-WAVE. Allows data to be transferred to and from PV-WAVE via bi-directional pipes and PV-WAVE's standard I/O facilities.
waveinit, wavecmdwaveterm	Yes	No	Routines that allow a C or FORTRAN program to start PV-WAVE, execute commands, and exit PV-WAVE. No data is transferred back to the calling program. Available on UNIX systems only.
LINKNLOAD	Yes *	Yes	A system routine that allows PV-WAVE to call an external function via dynamic linked libraries. It is the simplest method for calling your own C code from PV-WAVE. Allows the transfer of binary data. Data is transferred between the C program and PV-WAVE via the wavevars routine.

Method	UNIX	OpenVMS	Use (Table continued)
cwavec	Yes	Yes	A routine that allows a statically linked C program to access PV-WAVE. Data is transferred between the C program and PV-WAVE via the wavevars routine.
cwavefor	Yes	Yes	Works like cwavec, except from a statically linked FORTRAN program.
Option Programming Interface (OPI)	Yes	Yes	For developers who want to create optional modules that can be loaded explicitly by any PV-WAVE user. These optional modules can be written in C or FORTRAN, and can contain new system functions or other primitives.
CALL_UNIX	Yes	No	A system routine that uses Remote Procedure Call (RPC) technology to allow PV-WAVE to call a separate application across a UNIX network.
CALL_WAVE	Yes	No	A system routine that uses Remote Procedure Call (RPC) technology to allow a separate application to call PV-WAVE across a UNIX network.
Socket OPI	Yes	Yes	Allows you to treat network connections as streams of bytes that can be read from or written to. With the Socket OPI, you can write client and server applications entirely in PV-WAVE.

* Not currently available for all versions of UNIX.

The following table summarizes the methods of interapplication communication that can be used with Windows.

Method	Use
LINKNLOAD	A system routine that allows PV-WAVE to call an external function via dynamic linked libraries. It is the simplest method for calling your own C code from PV-WAVE. Allows the transfer of binary data.
cwavec	A routine that allows a dynamically linked C program to access PV-WAVE. Data is transferred between the C program and PV-WAVE via the wavevars routine (or PV-WAVE variable handles if you use the Option Programming Interface).
cwavefor	A routine that allows a dynamically linked FORTRAN program to access PV-WAVE. Data is transferred between the FORTRAN program and PV-WAVE via the wavevars routine.
DDE	Dynamic Data Exchange functions allow client applications to call PV-WAVE and modify variables, query variables, and execute functions and procedures.

Method Use (Table continued)

Option Programming Interface (OPI) For developers who want to create optional modules that can be loaded explicitly by any PV-WAVE user. These optional modules can be written in C or FORTRAN, and can contain new system functions or other primitives.

Socket OPI Allows you to treat network connections as streams of bytes that can be read from or written to. With the Socket OPI, you can write client and server applications entirely in PV-WAVE.

The remainder of this document is a Quick Reference Guide. It allows you to easily locate information about PV-WAVE.

PV-WAVE Foundation/Advantage Quick Reference Guide

- or -

“Everything You Wanted To Know About PV-WAVE But Were Afraid To Open The Manuals”

Mike Mayer, Visual Numerics, Inc.

Last updated: May 1, 2001
PV-WAVE Version 7.01

Introduction

This quick reference guide contains all lists and tables, as well as all keywords and system variables from every PV-WAVE Foundation manual. The optional product guides (such as IMSL Mathematics Reference, IMSL Statistics Reference, Image Processing Toolkit, Signal Processing Toolkit, etc.) are not included. Subjects have been arranged alphabetically.

Some of the more commonly used functions and procedure have also been included. However, for a complete quick reference listing of the entire set of PV-WAVE Command Language functions and procedures, refer to *the PV-WAVE Reference, Volume 1*, Chapter 1 "Functional Summary of Routines", starting on page 1. That list of commands and their descriptions are arranged into functional groups such as Arrays, Graphics and Plotting, GUI Development, Programming, etc.

For a quick reference listing of the complete set of PV-WAVE Advantage advanced mathematics routines, refer to the *PV-WAVE IMSL Mathematics Reference*, Appendix B "Summary of Routines", starting on page B-1.

In this document, references are made to the various PV-WAVE manuals (hardcopy and online versions), which direct the reader to more detailed information. The manual references in this quick reference guide are abbreviated the same way as those used in the index of each manual:

<u>Abbreviation</u>	<u>Manual Name</u>
UG	<i>PV-WAVE User's Guide</i>
PG	<i>PV-WAVE Programmer's Guide</i>
R1	<i>PV-WAVE Reference Volume 1</i>
R2	<i>PV-WAVE Reference Volume 2</i>
R3	<i>PV-WAVE Reference Volume 3</i>
GUI	<i>PV-WAVE GUI Application Developer's Guide</i>

For example, "PG-304" directs the reader to reference the *Programmer's Guide*, page 304. In this case, it's the section Variable Handling Examples.

This quick reference guide refers to just the basic PV-WAVE Foundation product. However, the functionality described herein generally applies to other optional modules since they share the same language, and are built as extensions to the Foundation product. No direct references are made to the optional PV-WAVE products in this document.

Table of Contents

<u>Sec.</u>	<u>Subject</u>
1.	Algebraic Operators (numeric, relational, Boolean)
2.	Axis Style Options
3.	C Format Strings
4.	Clipping
5.	Color tables
6.	Constants (Defining Byte, Integer, Long, Float, String)
7.	Coordinate Systems (Data Display and Conversion)
8.	Data Types
9.	Date and Time
10.	Devices (Display and Hardcopy)
11.	Environment Variables (Unix) & System Logicals (VMS)
12.	Error Handling
13.	Executive Commands
14.	File Input/Output
15.	Fonts
16.	FORTRAN Format Codes
17.	General Graphics Routines
18.	Graphics/Plotting Routines (2D)
19.	Graphics/Plotting Routines (3D)
20.	Graphics/Plotting - Keywords and System Variables
21.	GUI Development Methods
22.	Image (Raster) Display
23.	Informational Commands
24.	Interapplication Communication Methods
25.	Libraries – Function and Procedure
26.	Linestyles
27.	Mapping
28.	Operating System and Environment Access
29.	Plot Symbols
30.	Program Execution Control
31.	Special Characters
32.	Subscript Ranges (Arrays and Matrices)
33.	System Variables
34.	Table Functions
35.	Variable Names and Reserved Words
36.	VDA Tools

1. Algebraic Operators (numeric, relational, Boolean)

Listed in order of operator precedence (1 = highest, 5 = lowest). Parentheses are used to force algebraic operator precedence. See PG-30.

Priority	Operator	Description
1	^	Caret -Exponentiation
2	*	Asterisk - Multiplication
2	#	Pound sign - Matrix multiplication
2	/	(Forward) slash - Division
2	MOD	Modulus or modulo
3	+	Plus sign - Addition
3	-	Minus sign - Subtraction, and unary minus/negation
3	<	Less-than sign - Comparative relational minimum
3	>	Greater-than sign - Comparative relational maximum
3	NOT	Boolean negation
4	EQ	Relational Equality
4	NE	Relational Not Equal
4	LE	Relational Less than or Equal
4	LT	Relational Less Than
4	GE	Relational Greater than or Equal
4	GT	Relational Greater Than
5	AND	Boolean AND (union)
5	OR	Boolean OR (intersection or inclusion)
5	XOR	Boolean eXclusive OR (mutual exclusivity)

2. Axis Style Options

Specify the axis styles with the [XYZ]Style/[XYZ].Style keyword/system variable. Each option is encoded in a bit. Bit values are additive and specified as a single number. See R2-526. Also see "Graphics/Plotting Routines (2D)" in this quick reference guide.

Bit	Value	Function
0	1	Exact. By default the end points of the axis are rounded in order to obtain even tick increments. Setting this bit inhibits rounding, making the axis fit the data range exactly.
1	2	Extend. If set, the axes are extended by 5% in each direction, leaving a border around the data.
2	4	None. If it is set, the axis and its text are not drawn.
3	8	No box. Normally, PLOT and CONTOUR draw a box style axis with the data window surrounded by axes. Setting this bit inhibits drawing the top and right axes.
4	16	Inhibits setting the Y-axis minimum value to zero, when the data are all positive and nonzero. The keyword Ynozero sets this bit temporarily.

3. C Format Strings

You can use C format strings in PV-WAVE with any of the functions that begin with the two letters "DC".

C-style Conversion Characters for Importing Data PG-A-20

Conversion Character	How the Data is Imported
c	Transfers character data, one character at a time.
e, f, g	Transfers double-precision floating-point data with optional sign, decimal point, and exponent. Precede with l for double precision.
d or i	Transfers a signed integer. Precede with l for long integer.
o	Transfers octal data.
x	Transfers hexadecimal data.
u	Transfers unsigned integer data.
s	Transfers character strings.
%	Used in pattern matching to produce a literal % symbol. No conversion occurs.

C-style Conversion Characters for Exporting Data See PG-A-21.

Conversion Character	How the Data is Exported
c	Transfers character data, one character at a time.
e or E	Transfers double-precision floating-point data using scientific (exponential) notation. Use the form [-]m.dxxxxx e ± xx or [-]m.dxxxxx ± Exx, where the number of d's is given by the precision.
f	Transfers double-precision floating point data in the form [-]m.dxxxxx, where the number of d's is given by the precision. Precede with l for double precision.
g or G	Uses %e or %f format, depending on the magnitude of the value being processed.
d or i	Transfer signed integer data.
o	Transfers octal data.
x or X	Transfers hexadecimal data.
u	Transfers unsigned integer data.
s	Transfers character strings.
%	Transfers a literal % symbol. No conversion occurs.

3. Clipping Controls in PV-WAVE

Command	Default Clipping	Clipping Options
CONTOUR (*), PLOT, SURFACE (*)	!P.Clip is set by these commands. It defines the default clipping rectangle, which is usually bounded by the coordinate axes.	Use the Clip keyword to specify a clipping rectangle within default clipping region. Set the NoClip keyword to override Clip and explicitly enforce the default condition. PClip is not a valid keyword for these commands. !P.Clip, and !P.Noclip are not recognized.
OPlot	The clipping rectangle is defined by the coordinate axes (the plot data region).	Use the Clip keyword to specify a clipping rectangle. Disable clipping altogether by setting /NoClip or !P.Noclip=1. If you disable clipping, then data that falls outside the region bounded by the coordinate axes is not clipped. PClip is not a valid keyword for OPlot.

Clipping Controls in PV-WAVE (cont.)

Command	Default Clipping	Clipping Options
PLOTS, XYOUTS, POLYFILL	Clipping is disabled	Use the Clip keyword to specify a clipping rectangle. Set the PClip keyword to override the default condition. PClip causes the value of !P.Clip to be used to define the clipping rectangle (usually the area bounded by the coordinate axes).

* Denotes a 3D Routine.

5. Color tables

PV-WAVE color tables, loaded via LOADCT (also see TVLCT). See R2-62.

Index	Color table	Index	Color table	Index	Color table
0	B-W Linear	6	Prism	12	16-Level
1	Blue/White	7	Red-Purple Linear	13	16-Level II
2	Grn-Red-Blu-Wht	8	Green/White	14	Steps
3	Red Temperature	9	Green/White Expntl'	15	Wave Special
4	Blue/Green/Red/Yellow	10	Green-Pink		
5	Std Gamma-II	11	Blue-Red		

List of colors loaded by the TEK_COLOR command into the first 32 indices of the current PV-WAVE color table (remaining color table is unchanged). See R3-9.

Ind	Color	Ind	Color	Ind	Color
0	Black	11	Blue-Cyan	22	Lavender
1	White	12	Blue-Magenta	23	Light Magenta
2	Red	13	Red-Magenta	24	Pale Green
3	Green	14	Dark Gray	25	Pale Turquoise
4	Blue	15	Light Gray	26	Dark Turquoise
5	Cyan	16	Pale Yellow	27	Light Blue
6	Magenta	17	Light Green	28	Medium Blue
7	Yellow	18	Cyan-Green	29	Blue-Purple
8	Orange	19	Turquoise	30	Purple
9	Green-Yellow	20	Dark Cyan	31	Light Purple
10	Green-Cyan	21	Blue-Gray		

6. Constants (Defining Byte, Integer, Long, Float, String)

Integer constants definitions and examples. Also see PG-20.

Base	Type	Form	Example
Decimal	Byte	nB	128B
	Integer	n	128
	Long	nL	128L
Hexadecimal	Byte	'n'XB	'2E'XB
	Integer	'n'X	'2E'X
	Long	'n'XL	'2E'XL
Octal	Byte	"nB	"43B
	Integer	"n	"43
	Long	'n'O	'377'O (Uppercase letter O)

Examples of valid floating point constants: 123.0, 123., 123.456, 0.123, .123, 123E5, 123.E-3.123E+12, 123.456E13. For double precision constants, append the character D to the value, or replace the scientific notation E with a D, e.g., 123.456789D, or 123.456789D+12.

For string constants, see PG-23. For non-printable characters, see PG-24.

<u>Printed string value</u>	<u>PV-WAVE syntax</u>
Hi there	'Hi there' or "Hi there"
Null string	" or "" (no space between quotes)
I'm happy	"I'm happy" or 'I'm happy'
"Double Quotes"	""Double Quotes""
'Single Quotes'	""Single Quotes""
1234	'1234' or "1234"
<ESC>[;H<ESC>[2J	\033[;H\033[2J' (Erase ANSI terminal)
<FF>	\014' (Form feed)
\ABC	\\ABC' (Print the backslash character)

7. Coordinate Systems (Data Display and Conversion)

PV-WAVE maintains three distinct coordinate systems for plotting and for retrieving coordinate information via the mouse. Also see UG-45.

<u>System</u>	<u>Usage</u>
DATA	Units are as they apply to the particular nature of the data being plotted (e.g., gallons, meters, volts, angstroms, frequency, velocity, pressure, inches, acceleration). Usually established after the first plot of the data.
DEVICE	Coordinate system of the hardware displaying the graphics; usually pixels. On many window systems the origin (0, 0) is in the lower left corner of the current PV-WAVE window.
NORMAL	On all display devices, regardless of size or aspect ratio, the left edge is 0.0, right edge is 1.0, bottom edge is 0.0, and the top edge is 1.0. On window systems this corresponds to the four edges of the current PV-WAVE window. On printers, it is the boundaries of the defined printable area. Commonly used for convenient "percentage" positioning of items displayed or printed to a device.

Spatial coordinate conversion utilities:

<u>Routine</u>	<u>Usage</u>
CONV_FROM_RECT	Convert from rectangular Cartesian coordinates (points) to polar, cylindrical, or spherical coordinates. R1-151.
CONV_TO_RECT	Convert from polar, cylindrical, or spherical coordinates to rectangular Cartesian coordinates (points). R1-156.

8. Data Types

The data type of a constant is determined by its syntax, as explained later in this section. In PV-WAVE there are eight basic data types, each with its own form of constant: See PG-19

<u>Type (code)</u>	<u>Description</u>
Byte	8-bit unsigned integers.
Integer	16-bit signed integers.
Longword	64-bit signed integers on Digital ALPHA UNIX platforms; 32-bit signed integers on all other platforms.

Data Types (cont.)

Type (code)	Description
Floating-Point	32-bit single-precision floating-point.
Double-Precision	64-bit double-precision floating-point.
Complex	Real-imaginary pair using single-precision floating-point.
Double Complex	Real-imaginary pair using double-precision floating-point.
String	Zero or more eight-bit characters that are interpreted as text.

In addition, structures are defined in terms of the eight basic data types. PG Chapter 6, *Working with Structures*, describes the use of structures in detail.

Type Conversion Functions See PG-33

Function	See Page	Description
STRING	R2-391 PG-174	Convert to string
BYTE	R1-90, PG-117, 184	Convert to byte
FIX	R1-351	Convert to integer
LONG	R2-74	Convert to longword integer
FLOAT	R1-354	Convert to floating point
DOUBLE	R1-292	Convert to double-precision Floating point
COMPLEX	R1-128	Convert to single-precision complex value
DCOMPLEX	R1-180	Convert to double-precision complex value

9. Date and Time

Date and time formats for STR_TO_DT function, and for transferring date/time data to/from ASCII files. The * represents a delimiter separating different fields. Also valid are slash (/), colon (:), hyphen(-), and comma (,). Also see UG-199-200.

Value	Template	Description	Examples
1	MM*DD*[YY]YY		09/05/94 (September 5, 1994)
2	DD*MM*[YY]YY		05-09-94 " "
3	ddd*[YY]YY		248,1994 " "
4	DD*mmm[mmmmmm]*[YY]YY		05/Sep/94 " "
5	[YY]YY*mmm*DD		1994-09-05 " "
-1	HH*Mn*SS[.SSS]		13:30:35.25 (1:30pm)
-2	HHMn		1330 " "

!DT Date/Time data structure, also see UG-196:

Element	Type	Valid Range
!DT.Year	Integer	0 to 999999
!DT.Month	Byte	1 to 12
!DT.Day	Byte	1 to 31
!DT.Hour	Byte	0 to 23
!DT.Minute	Byte	0 to 59
!DT.Second	Float	0.0000 to 59.9999
!DT.Julian	Double	Number of days calculated from Sept. 14, 1752. The decimal part contains the time as a fraction of a day.

!DT Date/Time data structure (cont.)

Element	Type	Valid Range
!DT.Recalc	Byte	Recalculation flag. Setting this flag to 1 forces the Julian day to be recalculated.

!PDT, main system variable for Date/Time plotting attributes. For more details, see list starting on R2-283:

Name	See Page	Description
!PDT.Box	R3-284	Enables/disables background box for date/time axis labels. Default is disabled.
!PDT.Compress	R3-284	Enables/disables compression.
!PDT.Exclude_Holiday	R3-284	Excludes holidays from results returned by date/time routines.
!PDT.Exclude_Weekend	R3-284	Excludes weekends from result returned by date/time routines.
!PDT.Max_Levels	R3-284	Sets maximum number of levels on a date/time axis.
!PDT.Month_Abbr	R3-284	Abbreviates month names based on space available on date/time axis.
!PDT.Start_Level	R3-285	Overrides starting level of date/time labels that PV-WAVE derives.
!PDT.DT_Crange	R3-285	Axis ranges generated by PLOT command.
!PDT.DT_Range	R3-285	Used to specify exact date/time axis range.
!PDT.DT_Offset	R3-285	Internal PV-WAVE use.
!PDT.Week_Boundary	R3-285	Allows setting day of week as boundary for numbering the start of the week on axis levels. 0=Sunday, 1 = Monday (default), etc.

Initial level of tick labels displayed on a Date/Time axis are set by the Start_Level keyword or !PDT.Start_Level system variable. Subsequent levels depend on data range and first level. For more detailed information, see the examples in "Creating Plots with Date/Time Data" beginning at UG-209 or in on-line help choose the Start_Level keyword for the procedure PLOT.

Index	Start Level	Index	Start Level
7	Year	2	Hour
6	Quarter	1	Minute
5	Month	0	Second
4	Week	-1	Auto-level
3	Day		

10. Devices (Display and Hardcopy)

Supported output device formats and devices. Select using the SET_PLOT command, or via WAVE_DEVICE environment variable/logical. See R2-B-1.

Device Name	See Page	Description
NULL	N/A	No graphic output
CGM	R3-B-6	Computer Graphics Metafile generator
HP	R3-B-8	Hewlett-Packard Graphics Language (HPGL) plotters
PCL	R3-B-14	Hewlett-Packard Printer Control Language (PCL)
PM	R3-B-18	Pixel Map output
PS	R3-B-20	PostScript devices

Devices (Display and Hardcopy) (cont.)

<u>Device Name</u>	<u>See Page</u>	<u>Description</u>
REGIS	R3-B-33	Regis graphics output devices
TEK	R3-B-36	Tektronix or compatible terminals
WIN32	R3-B-39	Microsoft Windows WIN32 driver
WMF	R3-B-53	Windows Metafile
X	R3-B-59	X Window System
Z	R3-B-85	Z-buffer device

!D device system variable structure. These only apply to the currently selected graphics device. Also see R3-272.

<u>Name</u>	<u>See Page</u>	<u>Description</u>
!D.Display_Depth	R3-272	Stores the current display depth.
!D.Fill_Dist	R3-272	The line interval, in device coordinates required to obtain a solid fill.
!D.Flags	R3-272	See detailed table below.
!D.N_Colors	R3-273	Number of simultaneously available colors.
!D.Name	R3-273	String containing the name of the device.
!D.Table_Size	R3-273	Number of color table indices available.
!D.Unit	R3-273	LUN of file open for output from current device.
!D.Window	R3-273	Index of currently open window.
!D.X_Ch_Size, !D.Y_Ch_Size	R3-273	Normal width and height of a character in device units.
!D.X_Px_Cm, !D.Y_Px_Cm	R3-273	Number of pixels per centimeter in the X and Y directions.
!D.X_Size, !D.Y_Size	R3-273	Total size of the display in the X any Y directions.
!D.X_Vsize, !D.Y_Vsize	R3-274	Reports the size of the current window, updated when window is resized.

!D.Flags (system variable). A longword of flags. Each bit is encoded as shown in this table (also see R3-272):

<u>Bit</u>	<u>Value</u>	<u>Function</u>
0	1	Device has scalable pixel size (e.g., PostScript).
1	2	Device can control line thickness with hardware.
2	4	Device can output text at an arbitrary angle using hardware.
3	8	Device can display images.
4	16	Device supports color.
5	32	Device can polygon fill with hardware.
6	64	Device hardware characters are monospace.
7	128	Device can read pixels (i.e., TVRD).
8	256	Device supports windows.

11. Environment Variables (Unix, Windows) & System Logicals (VMS). Also see Appendix B in *PV-WAVE Programmer's Guide*.

UNIX/OpenVMS

Under UNIX, PV-WAVE uses environment variables to determine its initial state. Under OpenVMS, logical names are used for the same purpose. Normally, you do not need to alter your environment.

<u>Name</u>	<u>See Page</u>	<u>Description</u>
WAVE_DEVICE	PG-B-1	Defines your terminal or window system.
WAVE_DIR	PG-B-2	Root of the PV-WAVE directory structure.
WAVE_PATH	PG-B-3	Sets the function and procedure library directory search path.
WAVE_STARTUP	PG-B-5	Points to the name of a command file that is executed when PV-WAVE starts.
WAVE_FEATURE_TYPE	PG-B-6	Sets the default operating mode to "runtime".
WAVE_RT_STARTUP	PG-B-6	Points to the name of a compiled procedure file that is executed when PV-WAVE initializes in runtime mode.

Windows

Under Windows, PV-WAVE obtains the information it needs to determine its initial state from environment variables. Normally, you do not need to alter your environment.

<u>Name</u>	<u>See Page</u>	<u>Description</u>
VNI_DIR	PG-B-13	Directory where PV-WAVE files are installed.
WAVE_DIR	PG-B-13	Top-level directory for PV-WAVE.
WAVE_DATA	PG-B-14	Directory where data is stored.
WAVE_DEVICE	PG-B-15	Defines your terminal or window system.
WAVE_STARTUP	PG-B-16	Points to the name of a command file that is executed when PV-WAVE starts.
WAVE_FEATURE_TYPE	PG-B-17	Sets the default operating mode to "runtime".
WAVE_RT_STARTUP	PG-B-18	Points to the name of a compiled procedure file that is executed when PV-WAVE initializes in runtime mode.

Other PV-WAVE Environment Variables (See PG-B-10)

WAVE_APPL	WAVE_ARL	WAVE_BIN	WAVE_CODEBOOK
WAVE_DEMO	WAVE_GALL	WAVE_GALL2	WAVE_GALL3
WAVE_HELPDIR	WAVE_HELP_PATH	WAVE_LANG	WAVE_LIB
WAVE_PATH	WAVE_RAY	WAVE_USER	WAVE_VERSION
LM_LICENSE_DIR			

12. Error Handling

Routines for handling execution errors. Also see PG-237.

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
BREAKPOINT	R1-81	Insert and remove breakpoints in programs for debugging.
CHECK_MATH	R1-110	Math error handling.
FINITE	R1-350	Check validity of floating point or double precision operands.
INFO	R2-34	Display information on many aspects of the current PV-WAVE session.

Error Handling (cont.)

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
MESSAGE	R2-110	Use in programs to issue custom error messages.
ON_ERROR	R2-127	Regular program error handling.
ON_ERROR_GOTO	R2-136	Jump to a statement if an error occurs.
ON_IOERROR	R2-137	I/O error handling.
RETALL	R2-281	Typed from the WAVE prompt, used primarily to recover from errors in procedures and functions.
RETURN	R2-282	Return control to the caller.
STOP	R2-386	Stop execution of a running program or batch file, and return control to the interactive mode.

13. Executive Commands

<u>Command</u>	<u>See Page</u>	<u>Action</u>
.RUN	PG-222, R3-308	Compiles and possibly executes text from files or from the WAVE> prompt.
.RNEW	R3-307	Erases main program variables then executes like .RUN.
.CON	UG-12,26 R3-306	Continues execution of a stopped program.
.GO	R3-306	Executes previously compiled main program from the beginning of the program.
.STEP	R3-310	Executes a single statement. Can be abbreviated ".S".
.SKIP	R3-309	Skips over the next statement and then single steps.
.SIZE	PG-225 R3-309	Resizes code area and data area used to compile programs, in terms of bytes. Usage: .SIZE code_size data_size.
.LOCALS	PG-226 R3-306	Resizes data area in terms of local variables and common block symbols. Usage: .LOCALS local_vars common_symbols.
..LOCALS	PG-226	A compiler directive used in a program to dynamically add space for local variable and common block symbols at runtime. Like .LOCALS but not an executive command.

14. File Input/Output

Procedures that open files (See PG-134 and R2-138 (UNIX/OpenVMS) or R2-144 (Windows)):

<u>Procedure</u>	<u>Description</u>
OPENR	Open an existing file for input (reading) only.
OPENW	Opens a new file for input and output. In Unix, if the named file already exists, the previous contents are destroyed. Under VMS, a file with the same name and a higher version number is created.
OPENU	Opens an existing file for input and output.

Logical Unit Numbers (Also see PG-136):

<u>LUN</u>	<u>Purpose</u>
0	Standard input stream, usually the keyboard.
-1	Standard output stream, usually the screen.
-2	Standard error stream, usually the screen.
1-99	General use in programs, or interactively.
100-128	Reserved, managed by GET_LUN and FREE_LUN procedures.

Binary data transfer commands (See PG-147).

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
READU	R2-263	Read binary data from specified file unit.
WRITEU	R3-196	Write binary data to specified file unit.
DC_READ_8_BIT	R1-187	Reads an 8-bit image file.
DC_WRITE_8_BIT	R1-229	Writes 8-bit images to a file.
DC_READ_24_BIT	R1-189	Reads a 24-bit image file.
DC_WRITE_24_BIT	R1-231	Writes 24-bit image data to a file.
DC_READ_TIFF	R1-223	Reads a TIFF image file.
DC_WRITE_TIFF	R1-249	Write TIFF image data.
ASSOC	R1-53	Map an array structure to a data file. Provides efficient direct access to binary data.
GET_KBRD	R1-368	Read single characters from the keyboard.

ASCII data transfer commands (for a list of default output formats for these, see section 15, "FORTRAN Format Codes"), Also see PG-139:

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
PRINT	R2-239	Write ASCII data to the screen.
READ	R2-263	Read ASCII data from the keyboard.
PRINTF	R2-239	Write ASCII data to a specified file.
READF	R2-263	Read ASCII data from a specified file.
DC_READ_FREE	R1-210	Read free format ASCII files.
DC_WRITE_FREE	R1-242	Write ASCII data to an ASCII free format file.
DC_READ_FIXED	R1-196	Read fixed-format ASCII data.
DC_WRITE_FIXED	R1-235	Write fixed-format ASCII data to a file.

Other I/O related commands (See PG-200):

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
DC_ERROR_MSG	R1-184	Returns text string associated with negative status code generated by a "DC" data import/export function that does not complete successfully.
DC_OPTIONS	R1-185	Sets the error message reporting level for all "DC" import/export functions.
EMPTY	R1-314	Flushes graphics buffers to display device.

15. Fonts

List of PV-WAVE software fonts, specified as a font command in the text string to be drawn (usually at beginning of string). Also see UG-255. See R3-293 (Chapter 5) for printed examples of these fonts.

Font Cmd	Description	Font Cmd	Description
!3	Simplex Roman (default)	!12 (!W)	Simplex Script
!4	Simplex Greek	!13	Complex Script
!5	Duplex Roman	!14	Gothic Italian
!6	Complex Roman	!15	Gothic German
!7	Complex Greek	!16	Cyrillic
!8	Complex Italic	!17	Triplex Roman
!9 (!M)	Math and special characters	!18	Triplex Italic
!10	Special Characters	!20	Miscellaneous
!11 (!G)	Gothic English		

List of PV-WAVE text formatting commands, used similarly to the Font Commands above. Also see UG-254.

Format Cmd	Description
!A	Shift above the division line.
!B	Shift below the division line.
!C	Create multiple-line annotation (Carriage Return/Line Feed in the text).
!D	Shift down to first level subscript. Decrease character size by factor of 0.62.
!E	Shift up to exponent level. Decrease character size by factor of 0.44.
!I	Shift down to index level. Decrease character size by factor of 0.44.
!L	Shift down to second level subscript. Decrease character size by factor of 0.62.
!N	Shift back to normal level and original character size.
!R	Restore position. Current position is set from top of saved positions stack.
!S	Save position. Current position is saved on top of the saved positions stack.
!U	Shift to upper subscript level. Decrease character size by a factor of 0.62.
!!	Print the ! symbol.

16. FORTRAN Format Codes

The following table shows data conversion characters, which specify the type of data that is being transferred. See PG-A8. Detailed descriptions begin on PG-A10

FORTRAN Format Codes that Transfer Data

Conversion Character	How the Data is Transferred
[n]A[w]	Transfers character data. n is a number specifying the number of times to repeat the conversion. If n is not specified, the conversion is performed once. w is a number specifying the number of characters to transfer. If w is not specified, all the characters are transferred.
[n]D[w.d]	Transfers double-precision floating-point data. n is a number specifying the number of times to repeat the conversion. w specifies the number of characters in the external field, and d specifies the number of decimal positions.
[n]E[w.d]	Transfers single-precision floating-point data using scientific (exponential) notation. The options are the same as for the D conversion character.
[n]F[w.d]	Transfers single-precision floating-point data. The options are the same as for the D conversion character.

FORTRAN Format Codes that Transfer Data (cont.)

Conversion Character	How the Data is Transferred
[n]G[w.d]	Uses E or F conversion, depending on the magnitude of the value being processed. The options are the same as for the D conversion character.
[n]I[w] or [n]J[w.m]	Transfers integer data. n is a number specifying the number of times to repeat the conversion. w specifies the width of the field in characters. m specifies the minimum number of non-blank digits required.
[n]O[w] or [n]O[w.m]	Transfers octal data. The options are the same as for the I conversion character.
[n]Z[w] or [n]Z[w.m]	Transfers hexadecimal data. The options are the same as for the I conversion character.
Q	Obtains the number of characters in the input record to be transferred during a read operation. This conversion character is used for input only. In an output statement, the Q format code has no effect except that the corresponding I/O list element is skipped.

NOTE Characters in square brackets [] are optional.

FORTRAN Format Codes that Do Not Transfer Data

The following table shows specifiers that are used only to specify the appearance of data, such as the number of spaces between values in a file. See PG-A-9. More detailed descriptions begin on PG-A10.

Specifier	Appearance of Transferred Data
:	The colon format code in a format string terminates format processing if no more items remain in the argument list. It has no effect if variables still remain on the list.
\$	During input, the \$ format code is ignored. During output, if a \$ format code is placed anywhere in the format string, the newline implied by the closing parenthesis of the format string is suppressed.
quoted string	During input, quoted strings are ignored. During output, the contents of the string are written out.
nH	FORTRAN-style Hollerith string, where n is the number of characters. Hollerith strings are treated exactly like quoted strings.
nX	Skips n character positions.
Tn	Tab. Sets the absolute character position n in the current record.
TLn	Tab Left. Specifies that the next character to be transferred to or from the current record is the nth character to the left of the current position.
TRn	Tab Right. Specifies that the next character to be transferred to or from the current record is the nth character to the right of the current position.

Default output formats used when writing data with PRINT, PRINTF, and DC_WRITE_FREE. Also see PG-155

Data Type	Format
Byte	I4
Integer	I8
Long Integer	I12
Float	G13.6
Double	G16.8

Default output formats used when writing data with PRINT, PRINTF, and DC_WRITE_FREE.
(cont.)

<u>Data Type</u>	<u>Format</u>
Complex	(', G13.6, ', G13.6, ')
String	A (alphanumeric character data)

17. General Graphics Routines

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
CURSOR	R1-167, UG-81	Read graphics cursor (mouse) position.
ERASE	R1-318	Erase the screen or current window.
PLOTS	R2-188	Plot only 2D/3D lines, points, and/or markers.
POLYFILL	R2-208, UG-57	Fill defined polygons with color or pattern.
TVCRS	R3-31	Set cursor position and visibility.
USERSYM	R3-51, UG-56	Define a symbol for plotting.
WINDOW	R3-183	Create a PV-WAVE drawing window.
XYOUTS	R3-229, UG-53	Display graphics text at a specified location.

18. Graphics/Plotting Routines (2D)

Two-dimensional plotting routines.

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
AXIS	R1-58, UG-67	Draw an annotated axis.
O PLOT	R2-149, UG-47	Overplot array over old axis.
PLOT	R2-175, UG-47	Plot array on new axis, set scaling.
PLOT_IO	R2-175	Plot with linear-log scaling.
PLOT_OI	R2-175	Plot with log-linear scaling.
PLOT_OO	R2-175	Plot with log-log scaling.
VEL	R3-63	Plot velocity vector field (random field).
VELOVECT	R3-66	Plot velocity vector field (regular field).

19. Graphics/Plotting Routines (3D)

Three-dimensional plotting/rendering routines. Also see UG-83.

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
CONTOUR	R1-137, UG-84	Draws contour plots.
CONTOUR2	R1-140, UG-84	Draws contour plots for scattered data.
IMAGE_CONT	R2-6, UG-89	Overlays contours onto image display of array.
POLYSHADE	R2-226	Draws shaded surface of connected 3D polygons.
RENDER	R2-276	Creates a ray-traced scene.
SHADE_SURF	R2-326, UG-117	Draws shaded 3D surface plots.
SHOW3	R2-342	Displays 2D array as a combination contour, surface, and image plot.
SURFACE	R2-422, UG-98	Draws 3D mesh surface plots.

20. Graphics/Plotting - Keywords and System Variables

Keywords preceded with a slash (e.g., /NoErase) are set to 1 (logical true), otherwise, they are set equal to some value(s) (e.g., Title='My Plot'). Keywords override their corresponding system variable. If keyword is not specified, the system variable takes effect. Also see R3-235.

NOTE: If a keyword in the following list also has an associated entry in the !P plotting structure, the !PDT Date/Time plotting structure, or the ![XYZ] axis structure it is noted in the "Description" column. Also see "Date and Time" in this quick reference.

<u>Keyword</u>	<u>See Page</u>	<u>Description</u>
Alignment	R3-235	Horizontal alignment of graphics text in relation to the displayed position in YOUTS. Can vary between 0.0 (flush left) and 1.0 (flush right).
Ax	R3-235	Angle of rotation about the display horizontal.
Az	R3-236	Angle of rotation about the Z-axis.
Background	R3-236 R3-276	Background color index of screen for ERASE. System variable: !P.Background
Bottom	R3-237	Color index of underside of mesh surface.
Box	R3-237 R3-284	Places a box around labels in a date/time axis. System variable: !PDT.Box
C_Annotation	R3-237	Sets labels drawn on each contour line.
C_Charsize	R3-238	Sets character size of contour labels.
C_Charthick	R3-238 R3-277	Sets thickness of contour label characters. System variable: !P.Charthick
C_Colors	R3-238	Vector or color indices for each contour.
C_Labels	R3-239	Specifies which contour levels should be labeled.
C_Linestyle	R3-240	Specifies the Linestyle used to draw each contour. See "Linestyles" in this quick reference guide.
C_Thick	R3-240	Specifies the line thickness of contour levels.
Channel	R3-241	Destination channel index of mask for operation.
Charsize	R3-241 R3-277	Sets overall character size for annotation. System variable: !P.Charsize
Charthick	R3-241 R3-277	Thickness of software font characters. System variable: !P.Charthick
Clip	R3-242 R3-277	Sets coordinates of a rectangle used to clip graphics output. See "Clipping" in this quick reference guide. System variable: !P.Clip
Color	R3-242 R3-277	Sets color index of text, lines, solid polygon fill, data, axes, and annotation. See "Colortables" in this quick reference guide. System variable: !P.Color
Compress	R3-243 R3-284	Compresses out weekends and holidays from a date/time axis. System variable: !PDT.Compress
Data	R3-244	Specifies the data coordinate system. See "Coordinate Systems" in this quick reference guide.
Device	R3-244	Specifies the device coordinate system. See "Coordinate Systems" in this quick reference guide.
DT_Range	R3-245 R3-285	Sets exact range of values in a date/time axis. System variable: !PDT.DT_Range
Fill_Pattern	R3-245	Hardware-dependent fill pattern index.
Follow	R3-245	Forces contour line-following drawing method.
Font	R3-246 R3-277	Specifies whether software or hardware fonts are used. See "Fonts" in this quick reference guide. System variable: !P.Font, R2-544.

Graphics/Plotting - Keywords and System Variables (cont.)

Keyword	See Page	Description
Gridstyle	R3-246	Sets linestyle of tick marks. See "Linestyles" in this quick reference guide.
Horizontal	R3-278	System variable: !P.Gridstyle
	R3-247	Draw mesh surface lines perpendicular to line of sight. Used with SURFACE routine.
Levels	R3-247	Vector containing contour levels to be drawn.
Line_Fill	R3-248	Fill polygons with parallel lines.
Linestyle	R3-248	Specifies linestyle to be used. See "Linestyles" in this quick reference guide.
	R3-278	System variable: !P.Linestyle.
Lower_Only	R3-249	Only underside of mesh surface is to be drawn.
Max_Levels	R3-249	Maximum number of levels on date/time axis.
	R3-284	System variable: !PDT.Max_Levels
Max_Value	R3-249	Data values equal to or above this value are ignored when contouring.
Month_Abbr	R3-249	Abbreviates month or quarter names to either three or one character(s), depending on available axis space.
	R3-284	System variable: !PDT.Month_Abbr
NLevels	R3-250	Number of equally spaced contour levels desired.
NoClip	R3-250	Suppresses clipping of plot, lines, and text,
	R3-280	System variable: !P.Noclip
NoData	R3-250	No data points are plotted, only axes and text.
NoErase	R3-251	Do not erase screen or page before plotting.
Normal	R3-251	Specifies the normalized coordinate system. See "Coordinate Systems" in this quick reference guide.
	R3-251	Number of data points to average when plotting.
Nsum	R3-251	Number of data points to average when plotting.
	R3-280	System variable: !P.Nsum
Orientation	R3-251	Angle in degrees counterclockwise from horizontal of the text baseline and lines used to fill polygons.
	R3-252	Indicates CONTOUR procedure is to overplot.
Overplot	R3-252	Indicates CONTOUR procedure is to overplot.
Path_Filename	R3-252	Name of a file to contain the contour positions.
Pattern	R3-253	Rectangular array of pixels giving the fill pattern.
PClip	R3-253	Forces PLOTS, POLYFILL, and XYOUTS to accept the value of !P.Clip. By default these routines ignore !P.Clip. See "Clipping" in this quick reference guide.
	R3-277	System variable: !P.Clip
Polar	R3-254	If set, produces polar plots with PLOT command.
Position	R3-254	Specification of the plot window.
	R3-280	System variable: !P.Position
Psym	R3-255	Specifies by reference number a symbol used to mark each data point. See "Plot Symbols" in this quick reference guide.
	R3-281	System variable: !P.Psym
!P.Multi	R3-279	Defines layout of multiple plots per page.
!P.Region	R3-282	Positioning of the plot and its associated annotation, provides margins around plot.
	R3-282	Contains the homogeneous 4-by-4 transformation matrix.
!P.T	R3-282	Contains the homogeneous 4-by-4 transformation matrix.
	R3-256	Saves 3D to 2D transformation matrix established by SURFACE and SHADE_SURF, and Ax/Az keywords, in system variable !P.T.
Save	R3-256	Saves 3D to 2D transformation matrix established by SURFACE and SHADE_SURF, and Ax/Az keywords, in system variable !P.T.
Size	R3-257	Character size as a factor of the normal size.
Skirt	R3-257	Draw mesh surface skirt to a given Z value.
Solid_Psym	R3-258	Symbols are drawn with solid lines no matter which linestyle is used to connect the symbols.

Graphics/Plotting - Keywords and System Variables (cont.)

Keyword	See Page	Description
Spacing	R3-258	Spacing between polygon fill lines.
Spline	R3-258	Interpolate contour paths using cubic splines.
Start_Level	R3-259	Initial level of tick labels to be displayed on a date/time axis. See "Date and Time" in this quick reference guide.
	R3-285	System variable: !PDT.Start_Level
Subtitle	R3-259	Produces subtitle under the X-axis.
	R3-282	System variable: !P.Subtitle
Symsize	R3-259	Size of symbols to be drawn when Psym is set. See "Plot Symbols" in this quick reference guide.
T3d	R3-260	Use generalized transformation matrix in !P.T.
	R3-282	System variable: !P.T3D
Text_Axes	R3-261	Plane of vector-drawn text in 3D.
Thick	R3-261	Thickness of lines connecting points in a plot.
	R3-283	System variable: !P.Thick
Tickformat	R3-261	FORTRAN-style format specifiers for tick labels on axes.
	R3-283	System variable: !P.Tickformat
Ticklen	R3-262	Length of the axis tick marks.
	R3-283	System variable: !P.Ticklen
Title	R3-262	Produces a main title centered above the plot.
	R3-283	System variable: !P.Title
Upper_Only	R3-262	Only upper part of mesh surface is to be drawn.
Week_Boundary	R3-263	Day of the week on which week tick marks are drawn for the week level on a date/time axis. See "Date and Time" in this quick reference guide.
	R3-285	System variable: !PDT.Week_Boundary
Width	R3-263	Returns width of text string in normalized coordinate units.
[XYZ]Axis	R3-263	Type of axis to be drawn by AXIS procedure.
[XYZ]Charsize	R3-264	Character size in axis annotation and its title.
	R3-286	System variable: ![XYZ].Charsize
![XYZ].Crange	R3-287	The <i>output</i> axis range. Setting this variable has no effect.
[XYZ]Gridstyle	R3-264	Changes the Linestyle of tick intervals on the x-, y-, and z-axes. See "Linestyles" in this quick reference guide.
	R3-288	System variable: ![XYZ].Gridstyle
[XYZ]Margin	R3-264	Specifies margins on sides of plot window.
	R3-288	System variable: ![XYZ].Margin
[XYZ]Minor	R3-265	Number of minor tick marks on the axis.
	R3-288	System variable: ![XYZ].Minor
[XYZ]Range	R3-265	Desired data range of the axis.
	R3-289	System variable: ![XYZ].Range
![XYZ].Region	R3-289	Contains the normalized coordinates of the region.
![XYZ].S	R3-289	Scaling factors for converting between data coordinates and normalized coordinates.
[XYZ]Style	R3-265	See "Axis Options" in this quick reference guide.
	R3-289	System variable: ![XYZ].Style
[XYZ]Tickformat	R3-266	Format specification for axis tick labels using FORTRAN style formats. See "FORTRAN Format Codes" in this quick reference guide.
	R3-291	System variable: ![XYZ].Tickformat
[XYZ]Ticklen	R3-266	Length of tick marks for each axis.
	R3-291	System variable: ![XYZ].Ticklen
[XYZ]Tickname	R3-267	Array of string text for each major tick mark.
	R3-291	System variable: ![XYZ].Tickname

Graphics/Plotting - Keywords and System Variables (cont.)

<u>Keyword</u>	<u>See Page</u>	<u>Description</u>
[XYZ]Ticks	R3-267 R3-292	Number of major tick intervals to draw for axis. System variable: ![XYZ].Ticks
[XYZ]Tickv	R3-267 R3-292	Data values for each tick mark. System variable: ![XYZ].Tickv
[XYZ]Title	R3-268 R3-292	Places a title on each axis. System variable: ![XYZ].Title
[XYZ]Type	R3-268	Type of axis to be drawn (linear, log, Julian).
![XYZ].Window	R3-292	Contains the normalized coordinates of the axis end points, the plot window. Changing its value has no effect.
YNoZero	R3-268	Inhibits setting the minimum Y axis value to zero when the Y data are all positive and nonzero.
Z	R3-268	Z coordinate if a Z parameter is not present. This is of use only if the 3D transformation is in effect.
ZValue	R3-269	Sets the Z coordinate, in normalized coordinates in the range 0 to 1, of the axis and data output from PLOT, OPLOT, and CONTOUR.

21. GUI Development

GUI Development Methods for PV-WAVE

<u>Method</u>	<u>Advantages</u>	<u>Disadvantages</u>
Widget Toolbox	Creates Motif or Windows GUIs. All programming done in PV-WAVE. All Motif widget classes are supported. Allows rapid prototyping. Highly flexible. No knowledge of C required.	Application developer must be experienced using the Motif Widget toolkit. portable between Motif Microsoft Windows environments.
WAVE Widgets	Easy to use. Creates Motif or Microsoft Windows GUIs. Completely portable between Motif and Microsoft Windows environments (no modification required). All programming done in PV-WAVE. No experience programming with the Motif widget toolkit required. No knowledge of C required. Users can create new and modified WAVE Widgets functions. Allows rapid prototyping. Contains sufficient functionality for most GUI development projects.	Limited number of widget classes are supported. Less overall flexibility in interface design.
VDA Tools	All programming done in PV-WAVE. Provides easy-to-call user interface components which can reduce overall programming time. Multiple instances of a VDA Tool are instantiated from the same source code. Provides code generation functions. Easy to internationalize (via a string server)	Requires substantial PV-WAVE programming experience to develop new VDA Tools not already provided. Less overall flexibility in interface design. Limited number of widget classes are supported (Comparable to WAVE Widgets)

GUI Development Methods for PV-WAVE (cont.)

<u>Method</u>	<u>Advantages</u>	<u>Disadvantages</u>
C-based applications that call PV-WAVE (The application interface can be developed in C, and, via PV-WAVE's interapplication communication functions, the C application can call PV-WAVE to perform data processing and display functions.)	Highly flexible. Can be C code generated by any GUI builder. Can be used to add Visual Data Analysis capability to an existing application.	Application developer must be experienced using the Motif or Windows widget toolkit. Most complicated method of application development. Knowledge of C required. Knowledge of interapplication communication required.

22. Image (Raster) Display

Display, animate, and read raster (pixel) images. Also see UG-121.

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
MOVIE	R2-116	Animation of a 3D array "stack" of images.
TV	R3-27	Raw raster image display, no scaling.
TVRD	R3-35	Read image or sub-image from raster display device.
TVSCL	R3-36	Raster image display. First byte scales images 0-255.
TVCRS	R3-31	Manipulates the image device cursor.
TVLCT	R3-33	Loads a user-defined color table into the display device.
LOADCT	R2-61	Loads a predefined color table into the display device.

23. Informational Commands

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
INFO	R2-34 PG-267	Prints information about variables, files, compiled procedures, memory usage, etc.
KEYWORD_SET	R2-48 PG-249	Test to see if a keyword parameter has been set.
N_ELEMENTS	R2-121 PG-250	Returns number of elements contained in any expression or variable.
N_PARAMS	R2-124 PG-249	Returns the number of non-keyword parameters used in calling a PV-WAVE procedure or function.
N_TAGS	R2-125 PG-103	Returns the number of structure tags contained in any expression.
SIZE	R2-351 PG-251	Returns detailed information about a variable.
TAG_NAMES	R3-4 PG-103	Returns a string array containing the names of the tags in a structure definition.

24. Interapplication Communication Methods

PV-WAVE provides the capability of calling external C or FORTRAN programs as well as having C or FORTRAN programs calling PV-WAVE. See Chapter 2: Interapplication Communication for UNIX and OpenVMS on GUI-15 and Chapter 3: Interapplication Communication for Windows on GUI-93 for more information.

The following table summarizes the methods of interapplication communication that can be used with UNIX or OpenVMS.

Method	UNIX	OpenVMS	Use
SPAWN	Yes	Yes	A system routine that executes an external program from within PV-WAVE. Allows data to be transferred to and from PV-WAVE via bi-directional pipes and PV-WAVE's standard I/O facilities.
waveinit, wavecmd, waveterm	Yes	No	Routines that allow a C or FORTRAN program to start PV-WAVE, execute commands, and exit PV-WAVE. No data is transferred back to the calling program. Available on UNIX systems only.
LINKNLOAD	Yes *	Yes	A system routine that allows PV-WAVE to call an external function via dynamic linked libraries. It is the simplest method for calling your own C code from PV-WAVE. Allows the transfer of binary data. Data is transferred between the C program and PV-WAVE via the wavevars routine.
cwavec	Yes	Yes	A routine that allows a statically linked C program to access PV-WAVE. Data is transferred between the C program and PV-WAVE via the wavevars routine.
cwavefor	Yes	Yes	Works like cwavec, except from a statically linked FORTRAN program.
Option Programming Interface (OPI)	Yes	Yes	For developers who want to create optional modules that can be loaded explicitly by any PV-WAVE user. These optional modules can be written in C or FORTRAN, and can contain new system functions or other primitives.
CALL_UNIX	Yes	No	A system routine that uses Remote Procedure Call (RPC) technology to allow PV-WAVE to call a separate application across a UNIX network.
CALL_WAVE	Yes	No	A system routine that uses Remote Procedure Call (RPC) technology to allow a separate application to call PV-WAVE across a UNIX network.

Method	UNIX	OpenVMS	Use (Table continued)
Socket OPI	Yes	Yes	Allows you to treat network connections as streams of bytes that can be read from or written to. With the Socket OPI, you can write client and server applications entirely in PV-WAVE.

* Not currently available for all versions of UNIX.

The following table summarizes the methods of interapplication communication that can be used with Windows.

Method	Use
LINKNLOAD	A system routine that allows PV-WAVE to call an external function via dynamic linked libraries. It is the simplest method for calling your own C code from PV-WAVE. Allows the transfer of binary data.
cwavec	A routine that allows a dynamically linked C program to access PV-WAVE. Data is transferred between the C program and PV-WAVE via the wavevars routine (or PV-WAVE variable handles if you use the Option Programming Interface).
cwavefor	A routine that allows a dynamically linked FORTRAN program to access PV-WAVE. Data is transferred between the FORTRAN program and PV-WAVE via the wavevars routine.
DDE	Dynamic Data Exchange functions allow client applications to call PV-WAVE and modify variables, query variables, and execute functions and procedures.
Option Programming Interface (OPI)	For developers who want to create optional modules that can be loaded explicitly by any PV-WAVE user. These optional modules can be written in C or FORTRAN, and can contain new system functions or other primitives.
Socket OPI	Allows you to treat network connections as streams of bytes that can be read from or written to. With the Socket OPI, you can write client and server applications entirely in PV-WAVE.

25. Libraries – Function and Procedure

Library	Description
std	Routines in the Standard library are fully tested and documented by Visual Numerics.
guitools	Contains an assortment of graphical user interface (GUI) routines. These routines perform color table modifications, display surface views, display and manipulate iso-surfaces, and provide access to a variety of other functions. The GUI routines all begin with Wg (e.g., WgSurfaceTool) and are described in the PV-WAVE Reference.
motif windows	Contains the standard WAVE Widgets routines for Motif and Microsoft Windows. For information on WAVE Widgets, see the PV-WAVE GUI Application Developer's Guide.

Libraries – Function and Procedure (cont.)

<u>Library</u>	<u>Description</u>
user	Users' library routines written and submitted by PV-WAVE users. This library contains such entries as routines for compressing images, making pie charts, creating 2D/3D bar graphs, and displaying 3D scatter plots. The routines in the Users' library are not documented in the PV-WAVE documentation set. For information on a routine in the user library, read the header of the .pro file for that routine.
vdatools	Contains routines used to build VDA Tools. These include the VDA Tools Manager (Tm) routines, VDA Utilities (Wo), and a set of prewritten VDA Tools (Wz).

26. Linestyles

Linestyles used for drawing lines are selected using C_Linestyle/none, Gridstyle!/P.Gridstyle, [XYZ]Gridstyle![XYZ].Gridstyle, Linestyle!/P.Linestyle keywords/system variables. Used by most plotting procedures. 0 (solid) is the default. See also R3-240, R3-278, R3-279, and R3-288.

<u>Index</u>	<u>X Windows Style</u>	<u>Windows Style</u>
0	Solid	Solid
1	Dotted	Short dashes
2	Dashed	Long dashes
3	Dash dot	Long-short dashes
4	Dash-dot-dot-dot	Long-short-short dashes
5	Long dashes	Long dashes

27. Mapping Commands

Two data sets are included with PV-WAVE for creating world and US maps: The World Databank II data set for global maps, and a data set based on the USGS Digital Line Graph data for U.S. maps. Additionally, there is a template for reading in your own map data set. See Chapter 12: Mapping with PV-WAVE at UG-289.

The PV-WAVE mapping functionality supports 17 different types of projections and provides the ability for you to define your own. See UG-295.

<u>Index</u>	<u>Projection</u>	<u>Index</u>	<u>Projection</u>
1	Equidistant Cylindrical	11	Oblique Azimuthal Equidistant Oblique
2	Lambert Conformal Conic	12	Polar Azimuthal Equidistant Oblique
3	Cylindrical Mercator	13	Polar Azimuthal Equal-Area
4	Sinusoidal	14	Oblique Azimuthal Equal-Area
5	Albers Equal-Area Conic	15	Transverse Mercator
6	Polyconic	16	Mollweide (Ellipsoid)
7	Polar Stereographic	99	Satellite (3D mapping onto a spheres)
8	Oblique Stereographic	-1	User-defined projection
9	Oblique Orthographic	0	No projection
10	Polar Orthographical		

The PV-WAVE mapping procedures include the following:

<u>Routine</u>	<u>See Page</u>	<u>Description:</u>
MAP	R2-83	Plots map data with a specified projection.
MAP_CONTOUR	R2-92	Plots contours on a map.
MAP_PLOTS	R2-95	Plots vectors or points on the current map projection.
MAP_POLYFILL	R2-97	Fills specified regions of a map.
MAP_REVERSE	R2-98	Converts X-Y coordinate data to longitude and latitude coordinates.
MAP_VELOVECT	R2-100	Plots a two-dimensional vector field on a map.
MAP_XYOUTS	R2-101	Adds text to a map
USGS_NAMES	R3-53	Queries a database of longitude/latitude coordinates for states, countries, cities and towns in the United States.

Methods for Optimizing Mapping Applications

<u>Optimization Method</u>	<u>Advantages</u>	<u>Disadvantages</u>
Subsetting data with MAP keywords Select, Range, Center, and Zoom.	Keywords are easy to use and accessible to all PV-WAVE users. Allow you to display only those portions of the map that are of interest. Reduces the overall amount of data to be plotted.	Subsetting can be a slow process for very large datasets. May not be sufficient if a large portion of the map must be displayed at once.
Reduce the number of map data points plotted with the Resolution keyword.	Useful for wide-area maps where fine detail may not be necessary. Reduces the overall amount of data to be plotted.	Cannot be used with the usgs_db dataset.
Use the File_Path keyword to save a subsetting dataset in a binary file that can be read and displayed quickly with the Read_Path keyword.	Subsequent calls to MAP can skip the steps of reading and subsetting the map data. Provides excellent performance for applications that make calls to the MAP procedure to plot and replot the same dataset.	Reduces the detail/accuracy of the original map.
Use DEVICE, /Copy or TVRD to create an image of a basemap that can be rapidly re-displayed.	Provides excellent performance.	Resolution is limited to the resolution of the window into which the data is copied.
Write a C or FORTRAN procedure to read and subset a	Useful for handling user-supplied datasets that are too large to read into memory in one chunk.	Only useful for C or FORTRAN programmers. Some knowledge of

28. Operating System and Environment Access

<u>Command</u>	<u>See Page</u>	<u>Description</u>
\$	UG-19	By itself from the WAVE> prompt, spawns a child process to the operating system prompt. If followed by an operating system command, executes it then immediately returns to WAVE> prompt. Not for use in programs.
SPAWN	R2-374 R2-377 GUI-7, 17	Spawns a child process to execute commands. Output generated by commands can be captured in a PV-WAVE variable. Can be used in programs (unlike \$).
GETENV	R1-366	Get value of an environment variable.
SETENV	R2-308	Set value of an environment variable.
setenv	PG-B2-5	UNIX operating system command
ENVIRONMENT	R1-315	Returns a string array containing all the environment strings for the PV-WAVE process. (UNIX/Windows)
SETLOG	R2-309	Set a VMS logical name.
TRNLOG	R3-25	Translate a VMS logical name to a PV-WAVE string.
DELLOG	R1-267	Delete a VMS logical name.
SET_SYMBOL	R2-319	Set a DCL interpreter symbol (VMS).
GET_SYMBOL	R1-373	Get a DCL interpreter symbol (VMS).
DELETE_SYMBOL	R1-264	Delete a DCL interpreter symbol (VMS).
CD	R1-99, GUI-13	Change current directory.
PUSHD	R2-248, GUI-13	Push directory to top of LIFO directory stack.
POPD	R2-237, GUI-13	Pop directory from top of LIFO directory stack.
PRINTD	R2-241, GUI-13	List all directories in LIFO (Last-In-First-Out) directory stack, and current directory.

29. Plot Symbols

Psym!/P.Psym (keyword/system variable) sets the plot-symbol index. Used by graphics routines PLOT, OPLOT, PLOTS, POLYFILL. Also see R3-255, R3-281.

<u>Value</u>	<u>Symbol Drawn</u>
0	No symbol, connect points with solid lines.
1	Plus sign (+)
2	Asterisk (*)
3	Period (dot/pixel)
4	Diamond
5	Triangle
6	Square
7	X
8	User-defined, see USERSYM procedure.
9	Home plate shape
10	Data points are plotted in the histogram mode, i.e., horizontal and vertical lines connect points in staircase fashion instead of straight lines directly between points.
11-50	See figure, R3-255, R3-281.
-value	Negative values connect symbols with solid lines.

30. Program Execution Control

Used to control program flow and execution. Also see PG-283.

<u>Routine</u>	<u>See Page</u>	<u>Description</u>
EXECUTE	R1-330	Compiles and executes PV-WAVE statement(s) contained in its string parameter during run-time.
EXIT	R1-333	Exits PV-WAVE.
HAK	R1-390	Hit Any Key. Pauses execution until key pressed.
STOP	R2-386	Stops execution of program, returns to prompt.
WAIT	R3-115	Suspends execution of PV-WAVE program for a specified period of time. 4

31. Special Characters

See R3-303.

<u>Character</u>	<u>Function</u>
ampersand (&)	The ampersand separates multiple statements on one line. Statements may be combined until the maximum line length of 511 characters is reached.
apostrophe (')	Delimits string literals and indicates part of an octal or hexadecimal constant.
asterisk (*)	In addition to denoting multiplication, designates an ending subscript range equal to the size of the dimension. For example, A(3:*) represents all elements of the vector A except the first three elements.
"at" sign (@)	When the "at" sign is the very first character in a PV-WAVE command line, it causes the compiler to substitute the contents of the command file whose name appears after @. In addition to searching the current directory for the file, PV-WAVE searches a list of locations where procedures are kept.
colon (:)	Ends label identifiers. Labels may only be referenced by GOTO and ON_ERROR statements. In addition, the colon is used in CASE statements. The colon also separates the starting and ending subscripts in subscript range specifiers. For example A(3:6) designates the fourth, fifth, sixth, and seventh elements of the variable A.
dollar sign (\$)	At the end of a line indicates that the current statement is continued on the following line. The dollar sign character may appear anywhere a space is legal except within a string constant (where it is interpreted literally). Any number of continuation lines are allowed. When the \$ character is entered as the first character after the PV-WAVE prompt, the rest of the line is sent to the operating system as a command.
exclamation point (!)	Begins the names of system-defined variables.

Special Characters (cont.)

Character	Function
period or decimal point (.)	Indicates in a numeric constant that the number is of floating-point or double-precision type. Example: 1.0 is a floating-point number. Also, in response to the WAVE> prompt, the period, if it is the first character on the line, begins an executive command. Also, the period precedes the name of a tag when referring to a field within a structure.
quotation mark (")	The quotation mark precedes octal numbers which are always integers and delimits string constants.
semicolon (;)	Begins a comment field of a statement. All text on a line following a semicolon is ignored by PV-WAVE.

32. Subscript Ranges (Arrays and Matrices)

Syntax for subsetting and accessing array/matrix elements. See PG-75.

Form	Meaning	Examples
E	A simple subscript expression	test_scores(5)
e0 : e1	Subscript range from e0 to e1	test_scores(5:8)
E : *	All points from element E to end	test_scores(5:*)
*	All points in the dimension	test_scores(*)
[e,f,g]	Access individual elements.	test_scores([5,8,12])

33. System Variables

See also PG-28, R3-271. Most are read/write, unless noted.

Name	See Page	Purpose
!C	R3-271	Cursor system variable. Subscript of element found by MIN or MAX.
!Century_Divider	R3-271	Help ensure that Y2K problems are avoided in older code.
!D	R3-272	Read-only information on current graphics output device.
!Date_Separator	R3-274	Default character used to separate the parts of a date string. Default is slash (/).
!Day_Names	R3-274	An array of strings containing the names of the days of the week.
!Dir	R3-274	Name of the main PV-WAVE directory.
!Display_Size	R3-274	Contains the pixel dimensions of the display screen.
!Dpi	R3-274	Read-only double precision value of pi. R2-540
!DT_Base	R3-274	Contains value of Julian Day 1 (September 14, 1752) as PV-WAVE date/time data structure.
!Dtor	R3-275	Read-only conversion from degrees to radians (pi/180).
!Edit_Input	R3-275	Enables/disables keyboard line editing.
!Err	R3-275	Contains code of last error message.
!Err_String	R3-275	Read-only, contains text of last I/O error message.
!Holiday_List	R3-275	Array of up to 50 date/time structures that represent holidays created by the procedure CREATE_HOLIDAY. No default value(s).

System Variables (cont.)

<u>Name</u>	<u>See Page</u>	<u>Purpose</u>
!Journal	R3-275	Read-only logical unit number of journal output, or 0.
!Lang	R3-275	Identifies the language currently being used. Default is "american".
!Month_Names	R3-275	Array of strings for the month names.
!Mouse	R3-276	Used by the CURSOR function to store the X and Y mouse position, mouse button status, and date/time stamp.
!Msg_Prefix	R3-276	Prefix string for error messages.
!Order	R3-276	Indicates direction of image transfer to screen; bottom-up (0), or top-down(1).
!P	R3-276	Contains plotting system variables. Also see "Graphics/Plotting Keywords and System Variables" in this quick reference guide.
!Path	R3-283	Contains the search path for procedures and functions.
!PDT	R3-283	Contains the system variables for plotting date/time axes. Also see section 9, "Date/Time" in this quick reference guide.
!Pi	R3-285	Read-only floating point value of pi.
!Prompt	R3-285	Contains PV-WAVE command prompt string.
!Quarter_Names	R3-286	An array of four strings, containing the names for fiscal quarters. The default values are 'Q1', 'Q2', 'Q3', and 'Q4'.
!Quiet	R3-286	Suppresses screen compiler messages.
!Radeg	R3-286	Read-only, converts radians to degrees.
!Start	R3-286	Contains the value of the date and time PV-WAVE was started in a date/time structure.
!Time_Separator	R3-286	Lets you change the default character (:) used to separate the parts of a time representation.
!Version	R3-286	Read-only, contains the current version number of PV_WAVE, operating system, architecture, and platform.
!Weekend_List	R3-287	Array of 7 long integers. Element 0 represents Sunday, and so on. The value of an element is 0 if the day is a weekday, and 1 if it is a weekend day or a holiday. This variable is built with the CREATE_WEEKENDS procedure. No default values.
!X, !Y, !Z	R3-287	Axis system variables for X, Y, and Z axes. Also see "Graphics/Plotting - Keywords and System Variables" in this quick reference guide

34. Table Functions

The table functions let you create tables and subset them in various ways. These functions are both powerful and easy to use. See page UG-227.

<u>Name</u>	<u>See Page</u>	<u>Purpose</u>
BUILD_TABLE	R1-85	Creates a new table from numeric or string vectors (one-dimensional arrays) of equal length.
QUERY_TABLE	R2-249	Lets you subset, rearrange, group, and sort table data. This function returns a new table containing the query results.
UNIQUE	R3-44	Removes duplicate elements from any vector (one-dimensional array).
GROUP_BY	R1-386	Performs summary (aggregate) functions to groups of rows in a PV-WAVE table variable.
ORDER_BY	R2-155	Sorts the rows in a PV-WAVE table variable to create a new table.

35. Variable Names and Reserved Words

Rules for defining PV-WAVE variables. Also see PG-27.

- Must begin with a letter and be 1 to 31 characters long.
- Second and subsequent characters may be a letter, digit, underscore character, or dollar sign.
- No spaces allowed. They are considered delimiters.
- Characters after the first 31 are allowed but ignored.
- Variable names are not case sensitive. Uppercase and lowercase characters are treated as the same.
- A variable may not have the same name as a PV-WAVE function or PV-WAVE reserved word.
- A syntax error will result. See the reserved word list below.

PV-WAVE reserved words. Do not use these words as variables, or as names for functions or procedures. See PG-27

AND	BEGIN	CASE	COMMON	DO	ELSE
END	ENDCASE	ENDELSE	ENDFOR	ENDIF	ENDREP
ENDWHILE	EQ	FOR	FUNCTION	GE	GOTO
GT	IF	LE	LT	MOD	NE
NOT	OF	ON_IOERROR	OR	PRO	REPEAT
THEN	UNTIL	WHILE	XOR		

36. VDA Tools

The Tools Manager API

The routines used for creating and manipulating VDA Tools are listed in the following table.

Functional Listing of Tools Manager General Routines (GUI-131)

<u>Types of Routines</u>	<u>Routine Name</u>
General Routines	TmDynamicDisplay
	TmEnumerateItems
	TmEnumerateToolNames
	TmGetMessage
	TmGetTop
	TmGetUniqueToolName
	TmInit
	TmRegister
	TmUnregister
	Attribute Routines
TmGetAttribute	
TmSetAttribute	
Method Routines	TmEnumerateMethods
	TmExecuteMethod
	TmGetMethod
	TmSetMethod

Functional Listing of Tools Manager General Routines (cont.)

<u>Types of Routines</u>	<u>See Page</u>	<u>Routine Name</u>
Variable Routines	GUI-421	TmAddVar
	GUI-425	TmDelVar
	GUI-431	TmDynamicShowVars
	GUI-438	TmEnumerateVars
	GUI-450	TmGetVarMainName
Save/Restore Routines	GUI-470	TmRestoreTemplate
	GUI-471	TmRestoreTools
	GUI-472	TmSaveTools
Intertool Communication Routines	GUI-419	TmAddSelectedVars
	GUI-427	TmDeselectVars
	GUI-436	TmEnumerateSelectedVars
	GUI-441	TmExport
	GUI-442	TmExportSelection
Code Generation Routines	GUI-422	TmCodeGen
	GUI-431	TmEndCodeGen
	GUI-476	TmStartCodeGen
List Routines	GUI-452	TmList Function
	GUI-453	TmListAppend Procedure
	GUI-454	TmListClear Procedure
	GUI-455	TmListDelete Procedure
	GUI-456	TmListDestroy Procedure
	GUI-457	TmListExtend Procedure
	GUI-458	TmListGetMethod Function
	GUI-461	TmListInsert Procedure
	GUI-463	TmListReplace Procedure
	GUI-464	TmListRetrieve Function
GUI-466	TmListSetMethod Procedure	

Graphical Element API Routines

The following table lists the routines used to manage graphical elements, which are primarily used to annotate plots -- lines, rectangles, text, axes, and legends.

Functional Listing of Tools Manager Graphical Routines (GUI-140)

<u>Types of Grael Routines</u>	<u>See Page</u>	<u>Grael Routine Name</u>
General Grael Routines	GUI-479	TmAddGrael
	GUI-481	TmAxis
	GUI-483	TmBitmap
	GUI-485	TmDelGrael
	GUI-488	TmEnumerateGraels
	GUI-492	TmGetGraelRectangle
	GUI-128,449	TmGetUniqueGraelName
	GUI-495	TmLegend
	GUI-496	TmLine
	GUI-498	TmRect
	GUI-500	TmSetGraelRectangle
	GUI-501	TmText

Functional Listing of Tools Manager Graphical Routines (cont.)

<u>Types of Grael Routines</u>	<u>See Page</u>	<u>Grael Routine Name</u>
Grael Method Routines	GUI-490	TmEnumerateGraelMethods
	GUI-490	TmExecuteGraelMethod
	GUI-491	TmGetGraelMethod
	GUI-499	TmSetGraelMethod
Grael Selection Routines	GUI-480	TmAddSelectedGrael
	GUI-486	TmDelSelectedGraels
	GUI-489	TmEnumerateSelectedGraels
Grael Z-Order Routines	GUI-484	TmBottomGrael
	GUI-503	TmTopGrael
Grael Grouping Routines	GUI-494	TmGroupGraels
	GUI-504	TmUngroupGraels
Grael Cut, Copy, Paste, and Delete Routines	GUI-423	TmCopy
	GUI-424	TmCut
	GUI-426	TmDelete
	GUI-467	TmPaste