





Visual Numerics, Inc.

Corporate Headquarters

Suite 400, 9990 Richmond Avenue Houston, Texas 77042 United States of America 713/784-3131

FAX: 713/781-9260

Boulder, Colorado

6230 Lookout Road Boulder, Colorado 80301 United States of America

303/530-9000 FAX: 303/530-9329

France

33-1-42-94-19-65 FAX: 33-1-42-94-94-22 33-1-34-51-26-26

FAX: 33-1-34-51-97-69

Germany

49-211-367-7122 FAX: 49-211-367-7100

Japan

81-3-5689-7550 FAX: 81-3-5689-7553

United Kingdom

44-0753-790-600 FAX: 44-0753-790-601

Copyright ©1993 by Visual Numerics, Inc.

The information contained in this document is subject to change without notice.

VISUAL NUMERICS, INC., MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Visual Numerics, Inc., shall not be liable for errors contained herein or for incidental, consequential, or other indirect damages in connection with the furnishing, performance, or use of this material.

All rights are reserved. No part of this document may be photocopied or reproduced without the prior written consent of Visual Numerics, Inc.

Restricted Rights Legend

Use, duplication or disclosure by the US Government is subject to restrictions as set forth in FAR 52.227-19, subparagraph (c)(I)(ii) of DOD FAR SUPP 252.227-7013, or the equivalent government clause for other agencies.

Restricted Rights Notice: The version of PV-WAVE described in this document is sold under a per-machine license agreement. Its use, duplication, and disclosure are subject to the restrictions in the license agreement.

Contents Summary

Volume 1

		•					
v.	re	t o	~	3		I.	ı
_		ıa	U	7	•	•	•

Chapter 1: Functional Summary of Routines 1

Chapter 2: Procedure and Function Reference (A–P) 29

Multivolume Index i

Volume 2

Chapter 2: Procedure and Function Reference (Continued: Q–Z) 1

Chapter 3: Graphics and Plotting Keywords 497

Chapter 4: System Variables 537

Chapter 5: Software Character Sets 563

Appendix A: Picture Index A-1

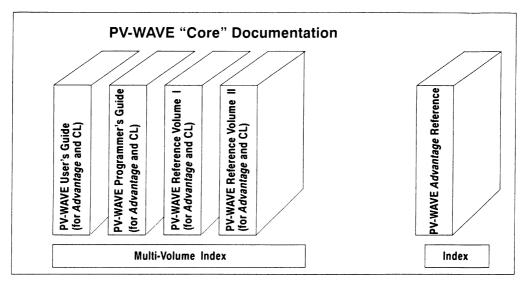
Multivolume Index i

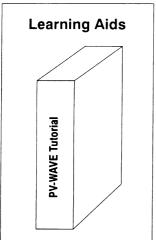
Preface

The *PV*-WAVE Reference is a two-volume set that describes the PV-WAVE Advantage and CL functions and procedures, keywords, and system variables.

This reference is part of a larger set of documentation; the entire set is shown in Figure I. Start with the *PV-WAVE Tutorial*, and then refer to this reference for details about the basic elements of the PV-WAVE Command Language. You will also want to refer to the *PV-WAVE User's Guide for Advantage and CL* and the *PV-WAVE Programmer's Guide for Advantage and CL* for detailed information.

For your convenience, all of the documents shown in Figure I are available online, as well as being available in a hardcopy format. Additional copies of the hardcopy documentation can also be ordered from Visual Numerics, Inc., by calling 800/447-7147.





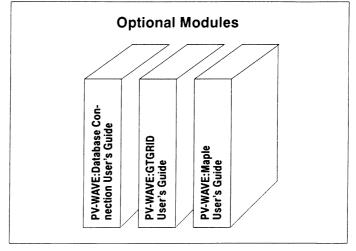


Figure I PV=WAVE documentation set; for more information about any one book you see shown here, refer to its preface, where the contents of each chapter are explained briefly. All documents are available both online and in a hardcopy format. Additional copies of the hardcopy documentation can be ordered by calling Visual Numerics, Inc., at 303/447-7147.

Contents of this Reference

Volume 1 Contains:

- **Preface** Describes the contents of this guide, lists the typographical conventions used, explains how to use the PV-WAVE documentation set, and explains how to obtain customer support.
- **Chapter 1: Functional Summary of Routines** A handy listing of PV-WAVE functions and procedures arranged into functional groups, such as image processing routines, input/ output routines, programming routines, and string processing routines. The basic syntax for each routine is also shown.
- Chapter 2: Procedure and Function Reference: A P Analphabetically arranged reference for all PV-WAVE procedures and functions. Most descriptions include one or more examples and cross references to related information.
- Appendix A: Picture Index A graphical index of the illustrations that appear in the PV-WAVE documentation set.
- **Index** A multivolume index that includes references to the PV-WAVE User's Guide, PV-WAVE Programmer's Guide, as well as both volumes of the Reference.

Volume 2 Contains:

- **Preface** Describes the contents of this guide, lists the typographical conventions used, explains how to use the PV-WAVE documentation set, and explains how to obtain customer support.
- Chapter 2 (Continued: Q Z) Procedure and Function Reference.
- Chapter 3: Graphics and Plotting Keywords Describes the keywords that can be used with the PV-WAVE graphics and plotting system routines.

- Chapter 4: System Variables Describes each of the PV-WAVE system variables.
- Chapter 5: Software Character Sets Shows each of the software character sets supported by PV-WAVE.
- Appendix A: Picture Index A graphical index of the illustrations that appear in the PV-WAVE documentation set.
- Index A multivolume index that includes references to the *PV*-WAVE User's Guide, *PV*-WAVE Programmer's Guide, as well as both volumes of the Reference.

Typographical Conventions

The following typographical conventions are used in this guide:

PV-WAVE code examples appear in this typeface.
 For example:

```
PLOT, temp, s02, Title='Air Quality'
```

• Code comments are shown in this typeface, below the commands they describe. For example:

```
PLOT, temp, s02, Title='Air Quality'
This command plots air temperature data vs. sulphur dioxide concentration.
```

Comments are used often in this reference to explain code fragments and examples. Note that in actual PV-WAVE code, all comment lines must be preceded by a semicolon (;).

- PV-WAVE commands are not case sensitive. In this reference, variables are shown in lowercase italics (*myvar*), function and procedure names are shown in uppercase (XYOUTS), keywords are shown in mixed case italic (*XTitle*), and system variables are shown in regular mixed case type (!Version). For better readability, all widget routines are shown in mixed case (WwMainMenu).
- A \$ at the end of a PV-WAVE line indicates that the current statement is continued on the following line. By convention,

use of the continuation character (\$) in this document reflects its syntactically correct use in PV-WAVE. This means, for instance, that strings are never split onto two lines without the addition of the string concatenation operator (+). For example, the following lines would produce an error if entered literally in PV-WAVE:

```
WAVE> PLOT, x, y, Title = 'Average $
   Air Temperatures by Two-Hour Periods'
        Note that the string is split onto two lines; an error message
        is displayed if you enter a string this way.
```

The correct way to enter these lines is:

```
WAVE> PLOT, x, y , Title = 'Average '+$
   'Air Temperatures by Two-Hour Periods'
       This is the correct way to split a string onto two command
```

The symbol means "or" when used in a usage line. It is not to be typed. For example, in the following command:

```
result = QUERY TABLE(table,
'[Distinct] * |col_i[alias][, ..., col_n[alias]]...
```

the | means use either * or col_i [alias] [, ..., col_n [alias]], but

Reserved words, such as FOR, IF, CASE, are always shown in uppercase.

Customer Support

If you have problems unlocking your software or running the license manager, you can talk to a Visual Numerics Customer Support Engineer. The Customer Support group researches and answers your questions about all Visual Numerics products.

Please be prepared to provide Customer Support with the following information when you call:

- The name and version number of the product. For example, PV-WAVE 4.2 or PV-WAVE P&C 2.0.
- Your license number, or reference number if you are an Evaluation site.
- The type of system on which the software is being run. For example, Sun-4, IBM RS/6000, HP 9000 Series 700.
- The operating system and version number. For example, SunOS 4.1.3.
- A detailed description of the problem.

The phone number for the Customer Support group is 303/530-5200.

Trademark Information

PostScript is a registered trademark of Adobe Systems, Inc.

QMS QUIC is a registered trademark of QMS, Inc.

HP Graphics Language, HP Printer Control Language, and HP LaserJet are trademarks of Hewlett-Packard Corporation.

Macintosh and PICT are registered trademarks of Apple Computer, Inc.

Open Windows and Sun Workstation are trademarks of Sun Microsystems, Inc.

TEKTRONIX 4510 Rasterizer is a registered trademark of Tektronix, Inc.

OPEN LOOK and UNIX are trademarks of UNIX System Laboratories, Inc.

PV-WAVE, PV-WAVE Command Language, PV-WAVE Advantage, and PV-WAVE P&C are trademarks of Visual Numerics, Inc.

OSF/Motif and Motif are trademarks of the Open Software Foundation, Inc.

X Window System is a trademark of the Massachusetts Institute of Technology

QUERY_TABLE Function

Subsets a table created with the BUILD_TABLE function.

Usage

```
result = QUERY_TABLE(table,
'[Distinct] * | col<sub>1</sub> [alias] [, ..., col<sub>n</sub> [alias]]
[Where cond]
[Group By colg_1 [,... colg_n]]
[Order By colo_1 [direction][, ..., colo_n [direction]]] ')
```

Note that the entire second parameter is a string and must be enclosed in quotes.

Input Parameters

table – The original table (created with the BUILD_TABLE function) on which the query is performed.

* — An optional wildcard character that includes *all* columns from the original table in the resulting table.

Distinct – A qualifier that removes duplicate rows from the resulting table.

 col_i — The list of columns that you want to appear in the resulting table. Use the asterisk (*) wildcard character to select all columns in the original table. The *col* names can be arguments to the calculation functions used with the Group By clause.

alias — Changes the input table's column name, col_i , to a new name in the output table. If no alias is specified, the input table's column name is used in the resulting table.

Where cond — A clause containing a conditional expression, cond, that is used to specify the rows to be placed in the resulting table. The expression can contain Boolean and/or relational operators.

Group By $colg_i$ – A clause specifying one or more columns by which the rows are grouped in the resulting table. Normally, the

grouped rows are data summaries containing results of calculation functions (Sum, Avg, etc.) applied to the columns. (Group By and Order By clauses are mutually exclusive: they cannot be used in the same function call.)

Order By $colo_i$ — Name of the column(s) to be sorted (ordered) in the resulting table. The first column named is sorted first. The second column named is sorted within the primary column, and so on. (Group By and Order By clauses are mutually exclusive: they cannot be used in the same function call.)

direction — Either Asc (the default) or Desc. Asc sorts the column in ascending order. Desc sorts the column in descending order. If neither are specified, the column is sorted in ascending order.

Returned Value

result — The resulting table, containing the columns specified by col_i, and the rows specified by the query qualifiers and clauses. If the query result is empty, and no syntax or other errors occurred, the result returned is -1.

Keywords

None.

Discussion

Before you can use QUERY_TABLE, you must create a table with the BUILD TABLE function. For details on BUILD TABLE, see the discussion of the BUILD_TABLE function in this chapter. See also Chapter 8, Creating and Querying Tables, in the PV-WAVE User's Guide.

A table query always produces a new table containing the query results, or -1 if the query is empty.

Any string or numeric constant used in a QUERY TABLE call can be passed into the function as a variable parameter. This means that you can use variables for numeric or string values in relational or Boolean expressions. For more information on passing parameters into QUERY_TABLE, see Passing Variable Parameters into Table Functions on page 279 of the PV-WAVE User's Guide.

Note

Within a QUERY_TABLE call, the Group By and Order By clauses are mutually exclusive. That is, you cannot place both Group By and Order By in the same QUERY_TABLE call.

Boolean and Relational Operators Used in Queries

The Where clause uses Boolean and relational operators to "filter" the rows of the table. You can specify any of the following conditions within a Where clause. Use parentheses to control the order of evaluation, if necessary.

- Comparison operators = (equal to), <> (not equal to),
 (less than), <= (less than or equal to), > (greater than),
 = (greater than or equal to)
- Compound search condition Not, And, Or
- Set membership test In

See the *Examples* section for more information on these operators.

You can also use PV-WAVE relational operators (EQ, GE, GT, LE, LT, and NE) in a Where clause instead of the SQL-style operators listed above.

Note //

When a literal string is used in a comparison, it must be enclosed in quotes—a different set of quotes than those used to delimit the entire QUERY_TABLE parameter string. For more information on using strings in comparisons, see *Using Strings in Where Clauses* on page 278 of the *PV-WAVE User's Guide*.

Calculation Functions Used with GROUP BY

The Group By clause is used in conjunction with calculation functions that operate on the values in the specified groupings. Each function takes one column name as its argument. See the *Examples* section for examples showing the use of calculation functions with Group By.

The calculation functions used with Group By are the following, where *col* is the name of a column:

- Avg(col) Averages the values that fall within a group.
- Count(col) Counts the number of occurrences of each data value that falls within a group.
- Max(col) Returns the maximum value that falls within a group.
- Min(col) Returns the minimum value that falls within a group.
- Sum(col) Returns the sum of the values that fall within a group.

Examples

For the following examples, assume table called phone_data contains information on company phone calls. This table contains eight columns of phone information: the date, time, duration of call, caller's initials, phone extension, cost of call, area code of call, and number of call.

The table used in these examples and the data used to create it are available to you. Enter the following command at the WAVE> prompt to restore the table and data:

```
WAVE> RESTORE, !dir+'/data/phone example.sav'
```

For more information on the structure of this table and more examples, see *Querying a Table* on page 270 of the *PV-WAVE User's Guide*.



For an example showing the use of the Distinct qualifier, see *Rearranging a Table* on page 271 of the *PV*-WAVE User's Guide.

The following examples show how to query this table in various ways using QUERY TABLE.

Create a new table containing only the phone extensions, area code, and phone number of each call made.

This example demonstrates a simple table query that produces a three-column subset of the original table.

```
new_table = QUERY TABLE(phone data, $
   'EXT, AREA, NUMBER')
```

A portion of the resulting table is organized as follows:

EXT	AREA	NUMBER
311	215	2155554242
358	303	5553869
320	214	2145559893
289	303	5555836
248	617	6175551999



For information on printing tables, see Formatting and Printing Tables on page 284 of the PV-WAVE User's Guide.

Example 2

Show me data on the calls that cost more than one dollar.

This example demonstrates how a Where clause is used to produce a subset of the original table, where all rows that contain a cost value of less than one dollar are filtered out.

```
new_tbl = QUERY_TABLE(phone_data, $
  '* Where COST > 1.0')
```

The following is an excerpt from the resulting table:

DATE	TIME	DUR	INIT	EXT	COST	AREA	NUMBER
901002	093200	21.40	TAC	311	5.78	215	2155554242
901002	094700	17.44	EBH	320	4.71	214	2145559893
901004	095000	3.77	DJC	331	1.02	512	5125551228

Show the total cost and duration of calls made from each phone extension for the period of time the data was collected.

This example demonstrates the use of the Group By clause. The column specified after Group By is the column by which the other specified columns are grouped. The calculation function Sum() is used to return the total cost and duration for each extension in the table.

The following command produces this result:

```
sum table = QUERY TABLE(phone data, $
   'EXT, SUM(COST), SUM(DUR) Group By EXT')
```

This produces the new table, called sum_table containing the columns EXT, SUM COST, and SUM DUR:

EXT	SUM_COST	SUM_DUR
0	0.00000	4.49000
248	0.350000	1.31000
289	0.00000	16.2300
311	5.78000	21.4000
320	4.71000	17.4400
331	1.02000	3.77000



The cost and duration columns are named in the result table, by default, with the prefix SUM. This prevents any confusion with the existing table columns that are already named COST and DUR. You can change these default names by including aliases in the QUERY TABLE function call.

The INFO command can be used to show the basic structure of this new table:

```
INFO, /Structure, sum_table
** Structure TABLE GB 2, 3 tags, 12 length:
               LONG
EXT
 SUM COST
               FLOAT
                           0.000000
                           4.49000
 SUM DUR
               FLOAT
```



The Structure keyword is used because tables are represented in PV-WAVE as an array of structures. For more information, see Tables and Structures on page 287 of the PV-WAVE User's Guide.

Example 4

Show me the extension, date, and total duration of all calls made from each extension on each date.

This example demonstrates a multiple Group By clause. For example, you can obtain a grouping by extension and by date. The result is a "grouping within a grouping".

The following command produces the desired result:

```
tbl = QUERY TABLE(phone data, $
   'EXT, DATE, Sum(DUR) Group By EXT, DATE')
```

	EXT	DATE	SUM_DUR
	0	901003	2.33000
	0	901004	2.16000
	248	901002	1.31000
	289	901002	16.2300
	311	901002	21.4000
	320	901002	17.4400
	331	901004	3.77000
	332	901003	2.53000
	358	901002	1.05000
	370	901003	0.450000
	370	901004	0.160000
	379	901003	1.53000
	379	901004	1.93000
	418	901003	0.350000
-			

Note that each multiple grouping produces one summary value. In this case the total duration is calculated for each extension/date grouping. For instance, in the table shown above, the row:

370	901003	0.450000

shows the total duration (0.450000) of all calls made from extension 370 on date 901003.

Example 5

Show the number of calls made from each extension for the period of time the data was collected.

This example demonstrates the Group By clause used with the Count function.

```
cost_sum = QUERY_TABLE(phone_data, $
   'EXT, Count(NUMBER) Group By EXT')
```

The result is a two-column table that contains each extension number and a count value. The count value represents the total number of times each extension number appears in the table.

EXT	COUNT_NUMBER
0	3
248	1
289	1
311	1
320	1
331	1
332	1
358	1
370	2
379	2
418	1



The parameter specified in the *Count* function has no real effect on the result, because the function is merely counting the number of data values in the primary column (that is, null values are not ignored). You can obtain the same result with:

```
cost_sum = QUERY_TABLE(phone_data, $
   'EXT, Count(DUR) Group By EXT')
```

Sort the phone data table by extension, in ascending order.

This example demonstrates how the Order By clause is used to sort a column in a table.

```
ext_sort = QUERY_TABLE(phone data, $
   '* Order By EXT')
```

Here is a portion of the resulting table.

	DATE	TIME	DUR	INIT	EXT	COST	AREA	NUMBER
90	1004	95300	1.360	JAT	0	0.00	303	5553200
90	1004	94700	0.800	JAT	0	0.00	303	5553200
90	1003	91600	2.330	JAT	0	0.00	303	5553440
90	1002	94800	1.310	RLD	248	0.35	617	6175551999
90	1002	94800	16.23	TDW	289	0.00	303	5555836
90	1002	93200	21.40	TAC	311	5.78	215	2155554242
90	1002	94700	17.44	EBH	320	4.71	214	2145559893

Example 7

Sort the phone_data table by extension, in ascending order, then by cost in descending order.

The table can be further refined by sorting the COST field as well.

```
cost_sort = QUERY_TABLE(phone_data, $
   '* Order By EXT, COST DESC')
```

This command produces a table organized like the previous table, except the COST column is now sorted in descending order within each group of extensions. The following illustrates the new table organization:

DATE	TIME	DUR	INIT	EXT	COST	AREA	NUMBER
901003	91600	0.450	MLK	370	0.12	212	2125557956
901004	95100	0.160	MLK	370	0.00	303	5551245
901004	94900	1.930	SRB	379	0.52	818	8185552880
901003	91600	1.530	SRB	379	0.41	212	2125556618

The In operator provides another means of filtering data in a table. This operator tests for membership in a set (one-dimensional array) of values. For example, the following array contains a subset of the initials found in the INIT column of the phone_data table:

```
nameset = ['TAC', 'KAR', 'OLL', 'ERD']
```

The following QUERY_TABLE call produces a new table that contains information only on the members of nameset:

```
res = QUERY_TABLE(phone_data, $
    ' * Where INIT In nameset')
```

See Also

UNIQUE, BUILD TABLE

For more information on QUERY_TABLE, see Chapter 8, Creating and Querying Tables, in the PV-WAVE User's Guide.

QUIT Procedure

Standard Library procedure that exits a PV-WAVE session.

Usage

QUIT

Parameters

None.

Keywords

None.

Discussion

QUIT simply prints a message and then calls the system routine EXIT.

See Also

EXIT

For more information, see Exiting PV-WAVE on page 13 of the PV-WAVE User's Guide.

RANDOMN Function

Returns one or more normally distributed floating-point pseudorandom numbers with a mean of zero and a standard deviation of 1.

Usage

 $result = RANDOMN(seed [, dim_1, ..., dim_n])$

Input Parameters

seed — A named variable containing the seed value for random number generation. The initial value of seed should be set to different values in order to obtain different random sequences. seed is updated by RANDOMN once for each random number generated. If seed is undefined, it is derived from the current system time.

 dim_i — The dimensions of the result. May be any scalar expression. Up to eight dimensions may be specified.

Returned Value

result — A single scalar or an array of the specified dimensions. Contains normally distributed floating-point pseudo-random numbers with a mean of zero and a standard deviation of 1. The floating-point numbers are in the range of -6.0 < x < 6.0.

Keywords

None.

See Also

RANDOMU

RANDOMU Function

Returns one or more uniformly distributed floating-point pseudorandom numbers over the range 0 < Y < 1.0.

Usage

 $result = RANDOMU(seed [, dim_1, ..., dim_n])$

Input Parameters

seed — A named variable containing the seed value for random number generation. seed is updated by RANDOMU once for each random number generated. The initial value of seed should be set to different values in order to obtain different random sequences. If seed is undefined, it is derived from the current system time.

 dim_i — The dimensions of the result. May be any scalar expression. Up to eight dimensions may be specified.

Returned Value

result — Returns a scalar or array of pseudo-random numbers over the range 0 < Y < 1.0. If no dimensions are specified, RANDOMU returns a scalar result.

Keywords

None.

Discussion

Uniform distribution means that, given a large enough sample of randomly-generated numbers, the same quantity of each number will be produced by RANDOMU. In other words, if you were to select a number generated by RANDOMU, you are as likely to pick any one number as another.

This example simulates the result of rolling two dice 10,000 times, and plots the distribution of the total using RANDOMU:

```
PLOT, HISTOGRAM(FIX(6 * RANDOMU(S, 10000)) + $
  FIX(6 * RANDOMU(S, 10000)) + 2)
```

In the above statement, the expression RANDOMU (S, 10000) is a 10,000-element floating-point array of random numbers greater or equal to 0 and less than 1. Multiplying this by 6 converts the range to 0 < Y < 6.

Applying the FIX function yields 10,000-point integer vectors from 0 to 5, one less than the numbers on one die. This is done twice, once for each die, and then 2 is added to obtain a vector from 2 to 12, the total of two die.

The HISTOGRAM function makes a vector in which each element contains the number of occurrences of dice rolls whose total is equal to the subscript of the element.

This vector is plotted by the PLOT procedure.

See Also

RANDOMN

RDPIX Procedure

Standard Library procedure that displays the X, Y, and pixel values at the location of the cursor in the image displayed in the currently active window.

Usage

```
RDPIX, image [, x0, y0]
```

Input Parameters

image — The image array loaded into the current window. May be any type. Pixel values are read from within the borders of the image; they are not read in relation to the full display area of the screen. This avoids scaling difficulties.

x0 – The X coordinate of the lower-left corner of the image displayed in the currently active window.

y0 – The Y coordinate of the lower-left corner of the image displayed in the currently active window.

Keywords

None.

Discussion

RDPIX runs on workstation displays only.

The X, Y, and pixel values under the cursor position are constantly displayed and updated. Pressing the left or center button makes a new line of output, saving the old line on the display. Pressing the right mouse button exits the procedure.

Example

```
OPENR, lun, !Data dir + 'mandril.img', $
   /Get lun
mandril img = BYTARR(512, 512)
```

- READU, lun, mandril_img

 Read in the PV•WAVE mandril demo image file.
- TV, mandril_img
 Display the image.
- RDPIX, mandril_img

Press the left or center mouse button to see pixel values, and the right mouse button to quit.

- TV, mandril_img, 100, 100

 Display the image offset on the screen by 100 pixels vertically and horizontally.
- RDPIX, mandril_img, 100, 100

 Read the pixel values from the offset image. Then press the left or center mouse button to see pixel values, and right mouse button to quit.

See Also

CURSOR

READ Procedures (READ, READF, READU)

Read input into PV-WAVE variables:

- READ reads ASCII (formatted) input from the standard input stream (PV-WAVE file unit 0).
- READF reads ASCII input from a specified file.
- READU reads binary (unformatted) input from a specified file. (No processing of any kind is done to the data.)

Usage

```
READ, var_1, \dots, var_n
READF, unit, var_1, \dots, var_n
READU, unit, var_1, \dots, var_n
```

Input Parameters

unit – The file unit from which the input will be taken.

 var_i — The named variables to receive the input.

Input Keywords

Format – (READ and READF only). Lets you specify the format of the input in precise detail, using a FORTRAN-style specification. FORTRAN-style formats are described in Appendix A, FORTRAN and C Format Strings, in the PV-WAVE Programmer's Guide.

If the Format keyword is not present, PV-WAVE uses its default rules for formatting the output. These rules are described in Table 8-7 on page 164 of the PV-WAVE User's Guide.

Discussion

READ and READF Procedures. If the *Format* keyword is not present and READ is called with more than one parameter, and the first parameter is a scalar string starting with the characters '\$(', this initial parameter is taken to be the format specification, just as if it had been specified via the *Format* keyword. This feature is maintained for compatibility with Version 1 of PV-WAVE.

READU Procedure. For nonstring variables, the number of bytes required for var_i is input. For string variables, PV-WAVE reads exactly the number of bytes contained in the existing string.

Example 1

This example reads a string from the standard input stream using the READ procedure. The value of the string is then displayed.

```
b = ' '
    Define a variable with string type.

READ, 'Enter a string: ', b
    Read a string from the terminal.

Enter a string: This is a string.

PRINT, b
    Display the contents of b.

This is a string.
```

Example 2

In this example, three integers are read from the standard input stream into a three-element integer array, nums, using READ. A file named readex.dat is then opened for writing, and the integers in nums are written to the file using PRINTF. The file is then closed. The Format keyword is used with PRINTF to specify the format of the integers in the file. The readex.dat file is then opened for reading, and the integers are read into a three-element integer array using READF with the Format keyword. The file is

then closed and the values that were read from readex.dat are displayed.

nums = INTARR(3)

Create a three-element integer array.

READ, 'Enter 3 integers: ', nums Read three integers from the standard input stream.

Enter 3 integers: 3 5 7

PRINT, nums

5

OPENW, unit, 'readex.dat', /Get Lun Open the readex.dat file for writing.

7

PRINTF, unit, nums, Format = '(3i1)' Write the integers to the file using a specified format.

FREE LUN, unit Close the file and free the file unit number.

OPENR, unit, 'readex.dat', /Get Lun Open the readex dat file for reading.

ints = INTARR(3)

Create a new three-element integer array.

READF, unit, ints, Format = '(3i1)' Read the three integers from the readex.dat file using the same specified format as when they were written.

PRINT, ints

Display the integers read from the file.

7 3 5

FREE LUN, unit

Close the file and free the file unit number.

In this example, READU is used to read an image of a galaxy from the file whirlpool.img, which is contained in the subdirectory data under the main PV-WAVE distribution directory.

```
OPENR, unit, FILEPATH('whirlpool.img', $
Subdir = 'data'), /Get_Lun
Open the file galaxy.dat for reading.

a = BYTARR(512, 512)
Create a byte array large enough to hold the galaxy image.

READU, unit, a
Read the image data.

FREE_LUN, unit
Close the file and free the file unit number.
```

See Also

OPENR, GET_LUN

For more information and examples, see *READU and WRITEU* on page 193 of the *PV*-*WAVE Programmer's Guide*.

For more information on format specification codes, see Appendix A, FORTRAN and C Format Strings, in the PV-WAVE Programmer's Guide.

REBIN Function

Returns a vector or array resized to the given dimensions.

Usage

 $result = REBIN(array, dim_1, ..., dim_n)$

Input Parameters

array – The array to be sampled. Cannot be of string or complex data type. Must have the same number of dimensions as the number of dimension parameters that you supply.

 dim_i — The dimension(s) of the resampled array. Must be integral multiples or factors of the original array's dimension(s).

Returned Value

result — The resized (resampled) vector or array.

Input Keywords

Sample – If present and nonzero, specifies that nearest neighbor sampling is to be used for both magnifying and shrinking operations.

If not present, specifies that bilinear interpolation is to be used for magnifying and that neighborhood averaging is to be used for shrinking. (Bilinear interpolation gives higher quality results, but requires more time.)

Discussion

The expansion or compression of each dimension is independent of the others; REBIN can expand or compress one dimension while leaving the others untouched.

This example creates an image of a shaded surface, resizes the image using REBIN, then displays the resized image.

```
x = DIST(20)
    Create a 20-by-20 single-precision, floating-point array where
    each element is proportional to its frequency.
SHADE SURF, x, Color = 0
    Display a shaded-surface representation of x.
LOADCT, 7
y = TVRD(0, 0, 640, 512)
    Get the content of the display subsystem's memory.
INFO, y
VARIABLE
                 BYTE
                                   = Array(640, 512)
z = REBIN(y, 320, 256)
    Resize the 640-by-512 array y and place the result in z.
WINDOW, 0, Xsize = 960, Ysize = 512
   Create window of size 960-by-512.
TV, y, 0
   Display larger image in position 0 of window.
TV, z, 5
   Display resized image in position 3 of window.
```

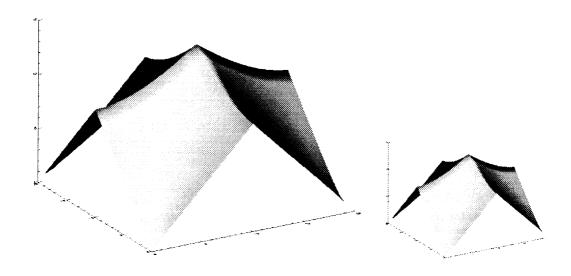


Figure 2-42 Original image (left); resized image (right).

See Also

CONGRID, REFORM

For more information on resampling and resizing images, see Image Magnification and Reduction on page 140 of the PV-WAVE User's Guide.

For information on interpolation methods, see Efficiency and Accuracy of Interpolation on page 170 of the PV-WAVE User's Guide.

REFORM Function

Reformats an array without changing its values numerically.

Usage

 $result = REFORM(array, dim_1, ..., dim_n)$

Input Parameters

array — The array that is to have its dimensions modified. Must have the same total number of elements as specified by the new dimensions.

 dim_i — The dimensions of the result.

Returned Value

result — The reformated array.

Keywords

None.

Discussion

REFORM returns a copy of *array* with the dimensions specified. If no dimensions are specified, then REFORM returns a copy of *array* with all dimensions of size 1 removed. Only the dimensions of *array* are changed; the actual data remains unaltered.



REFORM is useful for removing degenerate leading dimensions of size one. These leading dimensions can be created when you extract a subarray from an array with more dimensions.

Example

To use REFORM to remove degenerate leading dimensions:

```
a = intarr(10, 10, 10)
   a is a 3-dimensional array.
b = a(5, *, *)
   Extract a "slice" from a.
INFO, b, REFORM(b)
   Use INFO to show what REFORM did.
```

Executing the above statements produces:

```
В
            INT = Array(1, 10, 10)
<Expression> INT = Array(10, 10)
```

Note that the statements:

```
b = REFORM(a, 200, 5)
b = REFORM(a, [200, 5])
```

have identical effect. They create a new array b, with dimensions of (200,5), from a.

See Also

CONGRID, REBIN

REGRESS Function

Standard Library function that fits a curve to data using the multiple linear regression method.

Usage

result = REGRESS(x, y, wt [, yf, a0, sig, ft, r, rm, c])

Input Parameters

x — An array containing the independent values. Must be two-dimensional of size m by n, where m is the number of coefficients to be computed, and n is the number of data points.

y - A vector containing the dependent values. Must have n elements.

wt — A vector of weighting factors for determining the weighting of the multiple linear regression. Must have n elements.

Output Parameters

yf — An array containing the calculated values of y. It contains n number of elements

 a_0 – The constant term (offset) of the output function.

sig — A vector containing the standard deviations for the coefficients.

ft — The value of F in the standard F Test for the goodness of fit.

r – A vector containing the linear correlational coefficients.

rm – The multiple linear correlation coefficient.

c – The value of X^2 in the Chi-Squared test for the goodness of fit.

Returned Value

result — A column vector containing the coefficients (a_1 to a_m) of the function in x.

Keywords

None.

Discussion

REGRESS performs a multiple linear regression fit to a dataset with a specified function which is linear in the coefficients. The general function is given by:

$$f(x) = a_0 + a_1x_1 + a_2x_2 + ... a_mx_m$$

Weighting is useful when you want to correct for potential errors in the data you are fitting to a curve. The weighting factor, wt, adjusts the parameters of the curve so that the error at each point of the curve is minimized. For more information, see the section Weighting Factor on page 149, in Volume 1 of this reference.

Example

```
x = FLTARR(3, 9)
x(0, *) = [0., 1., 2., 3., 4., 10., 13., 17., 20.]
x(1, *) = [0.,3.,6.,9.,12.,15.,18.,19.,20.]
x(2, *) = [0., 4., 8., 12., 13., 14., 15., 18., 20.]
y = [5., 4., 3., 2., 2., 4., 5., 8., 9.]
   Create the data.
wt = FLTARR(9) + 1.0
coeff = REGRESS(x,y,wt,yf,a0,sig,ft,r,rm,c)
   Perform multiple linear regression with no weighting.
PLOT, yf, title='REGRESS EXAMPLE'
   Plot the fitted data.
PRINT, 'Fitted function:'
PRINT, ' f(x) = ', a0, ' + ', $
coeff(0, 0), 'x1 + ', $
coeff(0, 1), 'x2 + ', $
coeff(0, 2), 'x3'
PRINT, 'Standard deviations for ' +$
   'coefficients: ', sig
```

```
PRINT, 'F Test value:', ft
PRINT, 'Linear correlation coefficients: ', r
PRINT, 'Multiple linear correlation ' +$
    'coefficient: ', rm
PRINT, 'Chi-squared value: ', c
    Print all the output parameters.
```

See Also

CURVEFIT, GAUSSFIT, POLY_FIT, POLYFITW, SVDFIT

The REGRESS function is adapted from the program REGRES in *Data Reduction and Error Analysis for the Physical Sciences*, by Philip Bevington, McGraw-Hill, New York, 1969.

RENDER Function

Generates a ray-traced rendered image from one or more predefined objects.

Usage

```
result = RENDER(object_1, ..., object_n)
```

Input Parameters

*object*_i — A previously-defined object. Valid object types include CONE, CYLINDER, MESH, SPHERE, and VOLUME.

Returned Value

result — A 2D byte array (image) of size X-by-Y.

Input Keywords

Lights — A double-precision floating-point array defining the position and intensity (X,Y, Z, intensity) of all point light sources in the scene. It is of size 4-by-number_of_lights.

If this keyword is omitted, then a single light source is defined; this light source coincides with the automatically generated viewer's eye-point.

Sample — A long integer containing the number of randomly distributed rays to fire per pixel to perform anti-aliasing. The default is Sample=1.

Scale — If present, indicates that the resultant image should be scaled prior to conversion to bytes. By default, all generated shaded values are assumed to be in the range $\{0...1\}$ (see Discussion below).

Shadow – If present, indicates that shadow rays should be fired so that all points on all objects are not visible to all light sources. If not present, every point in a scene is visible to each light source.



For most visualization applications, you will want to omit the Shadows keyword, since this causes the ray tracer to run much faster.

Transform — A 4-by-4 double-precision floating-point array containing the local transformation matrix whose default is the identity matrix.

View — A 3-by-4 double-precision floating-point array used to override the auto-generation of the view to that specified. Uses the same format as is used for the *Info* keyword.

X — An integer defining the width of the byte image to be returned. Defaults to 256.

Y — An integer defining the height of the byte image to be returned. Defaults to 256.

Output Keywords

Info — A 3-by-4 double-precision floating-point array used to return the automatically calculated view as: [viewpoint, top_left_viewplane, bottom_left_viewplane, bottom_right_viewplane].

For example, you could define the variable k to contain this default view as follows:

```
k = DBLARR(3, 4)
RENDER(object, Info=k)
```

Discussion

RENDER generates an image from one or more objects using a technique called "ray tracing." The size of the returned byte image is X-by-Y, where X and Y each default to 256 unless overridden with the X and Y keywords. The returned image can be displayed using either the TV or TVSCL procedure.

Numerous objects can be rendered in the same scene. RENDER automatically generates the viewing information such that all objects are visible and the observer's viewpoint is on the positive Z axis looking towards the origin into the scene with a slight perspective. The *Transform* or the *View* keyword can be used to alter the default view. For more information, see *Setting Object and View Transformations* on page 201 of the *PV-WAVE User's Guide*.

The *Lights* keyword can be used to pass in an array of locations and intensities of point light sources. Except for the default light source (when none are specified by keyword), the light sources specified are not transformed. For best results, the sum of the intensities of light sources should equal 1.

The *Scale* keyword should be used in the following cases to ensure that all objects in the resulting image are in proportionate intensity:

- if the sum of the light source intensities is greater than 1, or
- if there exists a material property in the scene such that Color(i) * (Kamb(i) + Kdiff(i) + Ktran(i)) > 1.

If all of the values are less than 1, then the Scale keyword is not required, but you may wish to view the resultant image using TVSCL to improve the contrast.

By default, shadow rays are turned off and thus all points on all objects are visible to all lights. The firing of shadow rays can be turned on using the Shadows keyword.

Examples

```
TV, RENDER(SPHERE())
```

For more detailed examples, see Chapter 6, Advanced Rendering Techniques, in the PV-WAVE User's Guide.

See Also

CONE, CYLINDER, MESH, SPHERE, VOLUME

For more information, see Ray-tracing Rendering on page 196 of the PV-WAVE User's Guide.

For details on the SHADE VOLUME, TV, and TVSCL routines, see their descriptions in the PV-WAVE Reference.

REPLICATE Function

Forms an array with the given dimensions, filled with the specified scalar value.

Usage

 $result = REPLICATE(value, dim_1, ..., dim_n)$

Input Parameters

value — The scalar value used for filling the resulting array. May be of any scalar type, including scalar structures.

 dim_i — The dimensions of the result.

Returned Value

result — An array with the given dimensions, filled with the specified *value*. The resulting data type is that of *value*.

Keywords

None.

Example

This example uses REPLICATE to create a 4-by-3 string array. Each element of the array contains the string "string".

```
strs = REPLICATE("string", 4, 3)
INFO, strs
VARIABLE
               STRING
                             = Array(4, 3)
PRINT, strs
string
         string
                   string
                            string
string
         string
                   string
                            string
string
         string
                   string
                            string
```

See Also

MAKE_ARRAY

RESTORE Procedure

Restores the PV-WAVE objects saved in a file by the SAVE procedure.

Usage

RESTORE [, filename]

Input Parameters

filename – The name of the file from which the PV-WAVE objects should be restored. If not specified, the wavesave.dat file is used.

Input Keywords

Filename — The name of the file from which the PV-WAVE objects should be restored. If not present, wavesave.dat is used. This keyword serves exactly the same purpose as the filename parameter; only one of them needs to be provided.

Verbose — If present and nonzero, prints an informative message for each restored object.

Example

WAVE> SAVE, /All, Filename='mysave.dat' Save all local variables and system variables.

- User exits and then enters a new PV-WAVE session. -WAVE> RESTORE, 'mysave.dat' Restore all the saved variables.

See Also

SAVE, JOURNAL, COMPILE

For more information, see the section Using the RESTORE Procedure on page 40 of the PV-WAVE User's Guide.

RETALL Procedure

Issues RETURNs from nested routines. Used primarily to recover from errors in user-written procedures and functions.

Usage

RETALL

Parameters

None.

Keywords

None.

Discussion

RETALL issues RETURNs from nested procedures and functions until the main program level is reached. (The name RETALL is an abbreviation for RETurn ALL.)

When an error occurs in a procedure or function, control is left within that routine unless it contains special error handling instructions. Issuing the RETALL command causes control to return to the main level of PV-WAVE.



Typing RETALL will often make "disappearing variables" reappear.

See Also

RETURN, STOP

For more information, see *Error Handling in Procedures* on page 250 of the *PV*•*WAVE Programmer's Guide*.

RETURN Procedure

Returns control to the caller of a user-written procedure or function.

Usage

```
RETURN [, expr]
```

Input Parameters

expr — (Functions only). Returns the result of the function to the caller. Cannot be used in a procedure.

Keywords

None.

Example

```
FUNCTION SQUARE_IT, val
   x = val * val
   RETURN, x
        Return the value of x to the calling program.
```

See Also

RETALL, STOP

END

For more information about the role of the RETURN procedure, see Procedure or Function Calling Mechanism on page 248 of the PV-WAVE Programmer's Guide.

REVERSE Function

Standard Library function that reverses a vector or array for a given dimension.

Usage

result = REVERSE(array, dimension)

Input Parameters

array – The vector or array to be reversed.

dimension — (optional) The dimension of *array* to be reversed (the default is to reverse the first dimension).

Returned Value

result — The vector or array that has been reversed.

Keywords

None.

Discussion

REVERSE is helpful in a variety of applications; one example is its use in obtaining perfectly symmetrical figures, such as an hourglass, by simply creating a portion of the needed figure and then making a reverse image for the rest.



Running REVERSE is equivalent to running the ROTATE function with the correct parameter.

Example 1

This example exhibits the result of applying REVERSE to a 4-by-3 integer array.

```
a = INDGEN(4, 3)
```

Create a 4-by-3 integer array. Each element has a value equal to its one-dimensional subscript.

PRINT,	a				
0	1	2	3		
4	5	6	7		
8	9	10	11		
PRINT,	REVERS	SE(a)			
Reverse the rows of a.					
3	2	1	0		
7	6	5	4		
11	10	9	8		
PRINT, REVERSE(a, 2)					
Reverse the columns of a.					
8	9	10	11		
4	5	6	7		
0	1	2	3		
PRINT, REVERSE(REVERSE(a), 2)					
Reverse the columns and rows of a.					
11	10	9	8		
7	6	5	4		
3	2	1	0		

Example 2

The following commands first display the image contained in the PV-WAVE scientist3.dat file, and then rotate and redisplay it:

```
image1 = BYTARR(250, 200)
OPENR, 1, !Data_dir + 'scientist3.dat'
READU, 1, image1
TV, image1, 0
TV, REVERSE(image1), 1
```

See Also

ROTATE

REWIND Procedure

(VMS Only) Rewinds the tape on the designated PV-WAVE tape unit.

Usage

REWIND, unit

Input Parameters

unit — A number between 0 and 9 that specifies the magnetic tape unit to rewind. (Do not confuse this parameter with file logical unit numbers.)

Keywords

None.

See Also

For more information, see Accessing Magnetic Tape on page 229 of the PV-WAVE Programmer's Guide.

RGB_TO_HSV Procedure

Standard Library procedure that converts from the RGB color system to the HSV color system.

Usage

RGB_TO_HSV, red, green, blue, h, s, v

Input Parameters

red — Red color value(s). Can be scalar or vector, with values being short integers in the range 0 to 255.

green - Green color value(s). Must have the same number of elements as red.

blue - Blue color value(s).

Output Parameters

h — The Hue value. Must have the same number of elements as red and be in the range of 0 to 360.

s — The Saturation value. Must be in the range of 0 to 1.

v – The Value value. Must be in the range 0 to 1.

Keywords

None.

Discussion

RGB_TO_HSV converts colors from the RGB (Red, Green, Blue) color system to the HSV (Hue, Saturation, Value) color system.

See Also

HSV_TO_RGB, C_EDIT, COLOR_CONVERT, COLOR_EDIT, LOADCT, MODIFYCT, TVLCT, WgCeditTool, WgCtTool

For background information about color systems, see *Understanding Color Systems* on page 305 of the *PV*-WAVE User's Guide.

RM Procedure

Reads data into a one- or two-dimensional matrix.

Usage

RM, a [, rows, columns]

Input Parameters

rows - Number of rows in the matrix.

columns – Number of columns in the matrix.

Output Parameters

a - Named variable into which the data is stored.

Input Keywords

Complex – If present and nonzero, creates a complex matrix.

Double — If present and nonzero, creates a double-precision matrix.

Description

Procedure RM is used to read data into matrices according to the PV-WAVE matrix-storage scheme. If rows and columns are not specified, RM attempts to use the current definition of A to determine the number of rows and columns of A. If rows and columns are not specified and A is undefined or a scalar, an error is issued.

Upon invoking RM, the user is prompted with the row number for which input is expected. The prompt for the next row to be filled does not appear until the current row is filled. If the amount of data input for a particular row is larger than the defined number of columns of A, then the extra trailing input is ignored, and the prompt for the next row is given.

The matrix-printing procedures PM or PMF must be used to correctly print a matrix read in with RM.

Example 1: Reading a Simple Matrix

This example reads a 2×3 matrix and prints the results using the matrix-printing procedure PM.

```
RM, a, 2, 3
   Read a 2 x 3 matrix.
row 0: 11 22 33
row 1: 40 50 60
PM, a
   Output the matrix.
11.0000
               22.0000
                              33.0000
40.0000
               50.0000
                              60.0000
```

Example 2: Reading a Complex Matrix

In this example, a complex matrix is read. Notice that the elements input as integers are promoted to type complex with the value of the imaginary part set to zero.

```
RM, a, 3, 2, /Complex
Read the matrix; note that keyword Complex is set.

row 0: (1, 0) (1, 1)

row 1: 1 -1

row 2: (10.0, 0) (0, -10)

PM, a
Print the result.

( 1.00000, 0.00000)( 1.00000, 1.00000)
( 1.00000, 0.00000)( 0.00000, -10.0000)
( 10.0000, 0.00000)( 0.00000, -10.0000)
```

Example 3: Reading a Matrix to be Used with LUSOL

In this example, a 3×3 matrix and a 3×1 matrix are read. These matrices are then used in a call to the PV-WAVE Advantage function LUSOL.

```
RM, a, 3, 3
Read the coefficient matrix.

row 0: 1 3 3
row 1: 1 3 4
row 2: 1 4 3

RM, b, 3, 1
Read the right-hand side.

row 0: 1
row 1: 4
row 2: -1

x = LUSOL(b, a)
Call LUSOL to compute the solution.
```

```
PM, x
```

Output the results.

-2.00000

-2.00000

3.00000

PM, a#x - b

0.00000

0.00000

0.00000

See Also

PM, PMF, RMF

See Matrices on page 93 of the PV-WAVE Programmer's Guide for more information.

RMF Procedure

Reads data into a one- or two-dimensional matrix from a specified file unit.

Usage

RMF, unit, a [, rows, columns]

Input Parameters

unit — File unit from which the input is taken.

rows — Number of rows in the matrix.

columns — Number of columns in the matrix.

Output Parameters

a - Named variable into which the data is stored.

Input Keywords

Complex – If present and nonzero, creates a complex matrix.

Double — If present and nonzero, creates a double-precision matrix.

Format — Scalar string specifying the precise format of the data to be read. If Format is not specified, PV-WAVE uses its default rules for formatting the input. The character string should start with a left parenthesis and end with a right parenthesis. For example:

```
Format = '(f10.5)'
```

Description

RMF is used to read data from a specified file unit into a matrix according to the PV-WAVE matrix-storage mode. If rows and columns are not specified, RMF attempts to use the current definition of A to determine the number of rows and columns of A. If the arguments rows and columns are not specified and A is undefined or a scalar, an error is issued.

The matrix-printing procedures PM or PMF must be used to correctly print a matrix read in with RMF.

Example

This example reads in a 2×3 matrix from standard input (unit = 0) and prints the results using the matrix-printing procedure PM.

```
RMF, 0, a, 2, 3
Read matrix.

: 11 22 33
: 40 50 60

PM, a
Output the matrix.
```

11.0000	22.0000	33.0000
40.0000	50.0000	60.0000

See Also

PM, PMF, RMF

See Matrices on page 93 of the PV-WAVE Programmer's Guide for more information.

ROBERTS Function

Performs a Roberts edge enhancement of an image.

Usage

result = ROBERTS(image)

Input Parameters

image — The two-dimensional array that is to be enhanced.

Returned Value

result – A two-dimensional array of integer type containing the image that has been edge-enhanced.

Keywords

None.

Discussion

The ROBERTS function performs edge sharpening and isolation on *image*. It returns an approximation to the Roberts edge enhancement operator for images. This approximation is shown below:

$$G_{A}(j,k) = |F_{j,k} - F_{j+1,k+1}| + |F_{j,k+1} - F_{j+1,k}|$$

The resulting image returned by ROBERTS has the same dimensions as the input *image*.



Because the result image is saved in integer format, large original data values will cause overflow. Overflow occurs when the absolute value of the result is larger than 32,767.

Example

This example uses the ROBERTS function to apply the Roberts edge enhancement operator to an aerial image. The final edge enhanced image is the absolute value of the difference between the original image and the image resulting from the ROBERTS function.

OPENR, unit, FILEPATH('aerial_demo.img', \$
Subdir='data'), /Get_Lun

Open the file containing the image.

img = BYTARR(512, 512)

Create an array large enough to hold the image.

READU, unit, img
Read the image data.

FREE_LUN, unit

Close the file and free the file unit number.

WINDOW, 0, Xsize = 1024, Ysize = 512

Create a window large enough to hold two 512-by-512 images.

TV, img

Display the original image in the left half of the window.

HIST_EQUAL_CT, img

TV, ABS(img - ROBERTS(img)), 1

Display the absolute value of the difference between the original image and the result of the ROBERTS function in the right half of the window.



Figure 2-43 Original image (left), and Roberts edge enhanced image (right).

See Also

CONVOL, SHIFT, SOBEL

For background information, see Image Sharpening on page 160 of the PV-WAVE User's Guide.

ROT Function

Standard Library function that rotates and magnifies (or demagnifies) a two-dimensional array.

Usage

result = ROT(image, ang[, mag, xctr, yctr])

Input Parameters

image — The input image to be manipulated. Can be of any data type except string. Must be two-dimensional.

ang – The angle of rotation in degrees clockwise.

mag — The magnification or demagnification factor (see Discussion below).

xctr — The X subscript of the center of rotation. If omitted, xctr is equal to the number of columns in image divided by 2.

yctr – The Y subscript of the center of rotation. If omitted, yctr is equal to the number of rows in *image* divided by 2.

Returned Value

result - A rotated and magnified (or demagnified) image. The dimensions are the same as those for the input image.

Input Keywords

Interp — If present and nonzero, specifies that bilinear interpolation is to be used. Otherwise, the nearest neighbor interpolation method is used.

Missing — Data value to substitute for pixels in the output image that map outside the input image.

Discussion

ROT uses the POLY_2D function to rotate and scale the input image.

The magnification factor can be of integer or floating-point data type. It is specified as follows (with 1 being the default value):

mag = 1 no change

mag > 1 causes magnificationmag < 1 causes demagnification

For example, if *mag* is set to 0.5, this would result in an image half the size of the original image, and if *mag* is set to 3, this would result in an image three times the size of the original.



If you only need to rotate an image by 90-degree increments, the ROTATE function is more efficient to use.

If you need a more accurate bilinear interpolation method, use the ROT INT function.

Example

This example uses ROT to both rotate and demagnify an image. The image is rotated 135 degrees clockwise, while the demagnification factor is 0.63. Also, pixels that map outside the original image are assigned a value of 0.

```
OPENR, unit, FILEPATH('x2y2.dat', $
Subdir = 'data'), /Get_Lun
Open the file containing the image.

image = BYTARR(320, 256)
Create an array large enough to hold the image.

READU, unit, image
Read the image.
```

Close the file and free the file unit number.

FREE LUN, unit

WINDOW, 0, Xsize = 640, Ysize = 256

Create a window large enough to contain two 320-by-256 images.

TV, image, 0 Display the original image in the left half of the window.

a = ROT(image, 135, 0.63, Missing = 0)
Rotate the image 135 degrees clockwise, and demagnify it by a factor of 0.63. Also, pixels that map outside the original image are assigned a value of 0.

TV, a, 1
Display the rotated and demagnified image in the right half of the window.

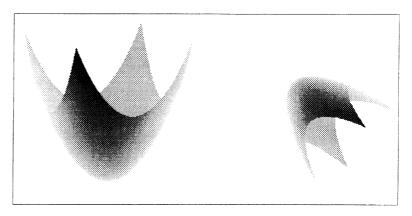


Figure 2-44 Original image (left); rotated/demagnified image (right).

See Also

POLY_2D, ROTATE, ROT_INT

For information on interpolation methods, see *Efficiency and Accuracy of Interpolation* on page 170 of the *PV*-WAVE User's Guide.

ROTATE Function

Returns a rotated and/or transposed copy of the input array.

Usage

result = ROTATE(array, direction)

Input Parameters

array - The array to be rotated. Must have either one or two dimensions.

direction - An integer that specifies the type of rotation to be performed, as shown below:

Direction	Transpose	Rotation Clockwise
0	No	None
1	No	90 °
2	No	180 °
3	No	270 °
4	Yes	None
5	Yes	90°
6	Yes	180°
7	Yes	270 °

direction is taken modulo 8, so a rotation of -1 is the same as 7, 9 is the same as 1, and so forth.

Returned Value

result — A copy of array that has been rotated and/or transposed by 90-degree increments.

Keywords

None.

Discussion

The resulting array is of the same data type as the input array. The dimensions of the result are the same as those of array if direction is equal to 0 or 2; the dimensions are switched if direction is 1 or 3.



To rotate by amounts other than multiples of 90 degrees, use the functions ROT and ROT_INT. However, note that ROTATE is more efficient than either of those functions.

Example 1

ROTATE may be used to reverse the order of elements in vectors. For example, to reverse the order of elements in the vector in variable X, use the expression:

```
ROTATE(X,2)

If

X = [0,1,2,3]

then

ROTATE(X,2) = [3,2,1,0]
```

Example 2

This example uses ROTATE to rotate an image of New York City by 90 degrees counterclockwise and displays the image both before and after the rotation.

```
OPENR, unit, FILEPATH('x2y2.dat', $
Subdir ='data'), /Get_Lun
Open the file containing an image of New York City.

img = BYTARR(320, 256)

READU, unit, img

FREE_LUN, unit
Create array to hold the image, read the image, and free the LUN.

WINDOW, 0, Xsize = 640, Ysize = 640
Create a window.
```

```
TV, img, 0, 321
TV, ROTATE(img, 1), 321, 257
TV, ROTATE(img, 2), 0, 0
TV, ROTATE(img, 3), 321, 0
   Display the image, before, and after rotations.
```

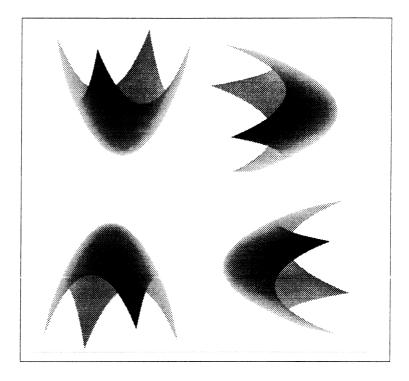


Figure 2-45 Original image (left) and rotations.

See Also

ROT, ROT_INT, REVERSE

ROT INT Function

Standard Library function that rotates and magnifies (or demagnifies) an image on the display screen.

Usage

result = ROT INT(image, ang [, mag, xctr, yctr])

Input Parameters

image — The input image to be manipulated. Can be of any data type except string. Must be two-dimensional.

ang – The angle of rotation in degrees clockwise.

mag — The magnification or demagnification factor (see Discussion below).

xctr — The X subscript of the center of rotation. If omitted, xctr is equal to the number of columns in image divided by 2.

yetr – The Y subscript of the center of rotation. If omitted, yetr is equal to the number of rows in *image* divided by 2.

Returned Value

result - A rotated and magnified (or demagnified) image. The dimensions are the same as those for the input *image*.

Keywords

None.

Discussion

ROT_INT calls the function POLY_2D to rotate and scale the input image.

The magnification factor can be of integer or floating-point data type. It is specified as follows (with 1 being the default value):

> no change mag = 1causes magnification mag > 1causes demagnification mag < 1

For example, if mag is set to 0.5, this would result in an image half the size of the original image, and if mag is set to 3, this would result in an image three times the size of the original.

ROT INT uses the bilinear interpolation method to rotate and scale the input image.



If a faster (but less accurate) method of interpolation is needed, use the related function ROT, which uses a nearest neighbor method.

See Also

ROT, POLY_2D, ROTATE

For information on interpolation methods, see Efficiency and Accuracy of Interpolation on page 170 of the PV-WAVE User's Guide.

SAVE Procedure

Saves variables or other specified objects in a file for later recovery by RESTORE.

Usage

SAVE
$$[, var_1, ..., var_n]$$

Input Parameters

 var_i — The named variables that are to be saved.

Input Keywords

All – If nonzero, specifies that everything (common blocks, system variables, local variables, compiled functions, and compiled procedures) should be saved.

Comm — If present and nonzero, causes all main level common block definitions to be saved.

Filename — The name of the file into which to save the PV-WAVE objects. If this keyword is not present, the file wavesave.dat is used.

Routines — If nonzero, saves all procedures and functions that are currently compiled in memory.

Variables — If present and nonzero, causes all current local variables to be saved.

Verbose — If present and nonzero, prints an informative message for each saved object.

XDR — If present and nonzero, causes the save file to be written in a portable format using XDR (eXternal Data Representation).

 Under UNIX, XDR is the only supported format, so specifying this keyword is unnecessary. Under VMS, XDR is used to interchange data with other versions of PV-WAVE. The default is a VAX-specific format that is more efficient to process.

See Also

JOURNAL, RESTORE, COMPILE

For more information, see Using the SAVE Procedure on page 40 of the PV-WAVE User's Guide.

SCALE3D Procedure

Standard Library procedure that scales a three-dimensional unit cube into the viewing area.

Usage

SCALE3D

Parameters

None.

Keywords

None.

Discussion

SCALE3D is useful for certain forms of perspective transformation, although it does not work for all forms.

SCALE3D does not use explicit parameters, but rather modifies the system variable !P.T for use as the implicit input and output parameters. Eight three-dimensional data points are created at the vertices of the three-dimensional unit cube. These eight points are transformed by !P.T. The system is translated to bring the minimum (X, Y, Z) point to the origin, and then scaled to make each coordinate's maximum value equal to 1.

Example

For an example, see *Procedure Used to Draw a House* on page 122 of the *PV*-WAVE User's Guide.

See Also

!P.T

SEC_TO_DT Function

Converts any number of seconds into PV-WAVE Date/Time values.

Usage

result = SEC_TO_DT(num_of_seconds)

Input Parameters

num_of_seconds — A scalar representing the number of seconds elapsed from the date specified in the system variable !DT_Base.

Returned Value

result — A PV-WAVE Date/Time variable containing the converted values.

Input Keywords

Base - A string containing a date, such as "3-27-92". This is the base date from which the number of seconds is calculated. The default value for Base is taken from the system variable !DT_Base.

Date Fmt - Specifies the format of the base date, if passed into the function. Possible values are 1, 2, 3, 4, or 5, as summarized in the following table:

Table 2-10: Valid Date Formats

Value	Format Description	Examples for May 1, 1992
1	MM*DD*[YY]YY	05/01/92
2	DD*MM*[YY]YY	01-05-92
3	ddd*[YY]YY	122,1992
4	DD*mmm[mmmmmm]*[YY]YY	01/May/92
5	[YY]YY*MM*DD	1992-05-01

where the asterisk (*) represents one of the following separators: dash (-), slash (/), comma (,), period (.), or colon (:).

For a detailed description of these formats, see Converting Your Data into Date/Time Data on page 229 of the PV-WAVE User's Guide.

Discussion

This function is useful for converting time stamps that count seconds to Date/Time values. Most time stamps count seconds from an arbitrary base date. For example, the UNIX time stamp counts seconds from January 1, 1970.

Example 1

This example shows how a value of 20 seconds is represented internally inside a PV-WAVE Date/Time variable after it has been converted with SEC_TO_DT. The example uses a base start date of January 1, 1970.

```
date = SEC_TO_DT(20, Base='1-1-70', $
  Date Fmt=1)
PRINT, date
   { 1970 1 1 0 0 20.0000 79367.000 0}
```

Example 2

Assume you have the following dataset which contains time stamps and associated measurements. The file contains data collected from January 1, 1990 to January 5, 1990. The base date for the clock is January 1, 1970.

```
6.3119520e+08 113
6.3128160e+08 768
6.3136800e+08 632
6.3145440e+08 227
6.3154080e+08 224
```

Assume the above file has been read into two PV-WAVE variables. The first column is read into a double-precision array called tarray. The second column is read into an array called fluid_level, which indicates the water level of a lake for each specified time period. You can convert the Date/Time data in Column 1 to PV-WAVE Date/Time data with the SEC_TO_DT function:

```
dtarray = SEC_TO_DT(tarray, Base='1-1-70')
PRINT, dtarray
{ 1990 1 1 12 0 0.00000 86672.500 0}
{ 1990 1 2 12 0 0.00000 86673.500 0}
{ 1990 1 3 12 0 0.00000 86674.500 0}
{ 1990 1 4 12 0 0.00000 86675.500 0}
{ 1990 1 5 12 0 0.00000 86676.500 0}
```

Notice the SEC_TO_DT function creates an array containing a PV-WAVE Date/Time structure for each of the seconds values. The Julian day shown for each Date/Time is automatically adjusted to reflect the PV-WAVE system base date of September 14, 1752.

See Also

```
DT_TO_SEC, STR_TO_DT, VAR_TO_DT, JUL_TO_DT, !DT_Base
```

For more information, see Chapter 7, Working with Date/Time Data, in the PV-WAVE User's Guide.

SELECT_READ LUN Procedure

Waits for input on any of a list of logical unit numbers.

Usage

SELECT READ LUN, luns

Input Parameters

luns — Vector of logical unit numbers.

Output Parameters

luns — Vector of logical unit numbers.

Input Keywords

Timeout — Amount of time, in seconds, this procedure should block while waiting for input. This may be a floating-point number. The default is to block forever.

Widget — If present and nonzero, selects the X11 socket.

Output Keywords

Widget — If there is no X11 input, then the value of this keyword is set to -1.

Description

This procedure checks for input on all the logical unit numbers specified in the luns vector. When it returns, those unit numbers without input are set to -1. By default, this procedure blocks forever waiting for input. Keyword Timeout can be used to reduce the time the procedure blocks. This procedure always returns when any one of the listed LUNs has input to be read.

This procedure is an interface to the select_read_lun C routine in the io.so shared library.

See Also

EXEC_ON_SELECT, ADD_EXEC_ON_SELECT, DROP_EXEC_ON_SELECT

SETDEMO Procedure

Standard Library procedure that defines key bindings and system variables to run the PV-WAVE demonstration system for the system on which PV-WAVE is started.

Usage

SETDEMO

Parameters

None.

Keywords

None.

Discussion

SETDEMO is called when PV-WAVE is started, and defines the key bindings and system variables used to run the demonstration system. The message that appears at the start of a session identifying the key bindings is written by this procedure.



Some key definitions may not be possible on all platforms.

SETDEMO is called by the PV-WAVE startup file:

- In UNIX, this file is /wave/bin/wavestartup
- In VMS, this file is WAVE DIR: [000000.bin]wavestartup.dat

During startup, SETDEMO sets up the key bindings for the PV-WAVE demonstration, accessing the Help system, outputting the PV-WAVE status, and creating a subprocess on your system. It then displays a message indicating that the key bindings were set.

You can use SETDEMO to customize your keys for commonly used commands (through the DEFINE_KEY routine), to change the message that is printed to the screen when you invoke PV-WAVE, or to change the default key bindings.

See Also

DEFINE_KEY, SETUP_KEYS

For more information on using a startup file, and defining function keys and environment parameters used by PV-WAVE, see Chapter 2, Getting Started, in the PV-WAVE User's Guide.

SETENV Procedure

(UNIX Only) Adds or changes an environment string in the process environment.

Usage

SETENV, environment_expr

Input Parameters

 $environment_expr$ — A scalar string containing an environment expression to be added to the environment.

Keywords

None.

Example

SETENV, 'SHELL=/bin/sh'

See Also

GETENV, ENVIRONMENT

SETLOG Procedure

(VMS Only) Defines a logical name.

Usage

SETLOG, logname, value

Input Parameters

logname - A scalar string containing the name of the logical to be defined.

value — A string giving the value to which logname will be set. If value is a string array, logname is defined as a multi-valued logical where each element of value defines one of the equivalence strings.

Input Keywords

Concealed — If set, RMS interprets the equivalence name as a device name.

Confine — If set, prevents logname from being copied from the PV-WAVE process to its spawned subprocesses.

No_Alias - If set, prevents logname from being duplicated in the same logical table at an outer access mode. If another logical name with the same name already exists at an outer access mode, it is deleted.

Table – A scalar string giving the name of the logical table from which to delete *logname*. If this keyword is not present, the table LNM\$PROCESS TABLE is used.

Terminal — If set, prevents further iterative logical name translation on the equivalence name from being performed when SETLOG attempts to translate *logname*.

See Also

DELETE_SYMBOL, DELLOG, GET_SYMBOL, SET SYMBOL, TRNLOG

For more information on logical names and access modes, see the *VAX/VMS DCL Dictionary*. Information oriented to C programmers can be found in the *VMS System Services Manual*.

SET_PLOT Procedure

Specifies the device type used by PV-WAVE graphics procedures.

Usage

SET_PLOT, device

Input Parameters

device — A scalar string giving the name of the device to use. This parameter is case-insensitive.

Keywords

copy — If present and nonzero, PV-WAVE's internal color table is copied into the device. This is the preferred method if you are displaying graphics and each color index is explicitly loaded.

interpolate — If present and nonzero, the color table of the new device is loaded by interpolating the old color table to span the new number of color indices. This method works best when displaying images with continuous color ranges.

Discussion

The names of available devices are given in Table A-1 and Table A-2 on page A-2 of the *PV*=*WAVE User's Guide*.

If the *Copy* keyword parameter is set, the color table copying is straightforward as long as both devices have the same number of

color indices. However, if the new device has more colors than the old device, some color indices will be invalid. If the new device has less colors than the old, not all the colors are saved.

Examples

```
SET PLOT, 'ps'
    Send output to a PostScript file.
SET_PLOT, 'cgm'
    Send output to a CGM file.
```

See Also

!D, DEVICE

For background information, see Appendix A, Output Devices and Window Systems, in the PV-WAVE User's Guide.

SET SCREEN Procedure

Standard Library procedure that establishes a new position for the rectangular plot area based on input values specified using the device coordinate system.

Usage

SET SCREEN, xmin, xmax [, ymin, ymax]

Input Parameters

xmin — The position of the left edge of the rectangular plot area in device coordinates.

xmax – The position of the right edge of the rectangular plot area in device coordinates.

ymin — The position of the bottom edge of the rectangular plot area in device coordinates.

ymax — The position of the top edge of the rectangular plot area in device coordinates.

Input Keywords

Cursor — Lets you set the region for the rectangular plot area interactively with the mouse:

- If nonzero, lets you set the boundaries by clicking the mouse at the corners of the rectangular plot area that you want to use. Click first to set the lower-left corner, then once again to set the upper-right corner. Any input parameters for SET_SCREEN are ignored.
- If zero, uses the input parameters as specified.

Region — If present, uses the system variable !P.Region to set the plot area. The default is to use !P.Position.

Discussion

SET_SCREEN is used to set up the area of the display that will be used for plotting. It is identical to the SET_VIEWPORT procedure, except that the input parameters are in device coordinates for SET SCREEN.

If only *xmin* and *xmax* are provided, the values for *ymin* and *ymax* are calculated. Calling SET_SCREEN with four input parameters is the same as setting the !P.Position system variable or using the *Position* keyword with the plotting commands

SET_SCREEN overrides the effect of the system variables !X.Margin and !Y.Margin.



To find out the current values of your device, type either: INFO, /Structure, !D
PRINT, !D.X Size, !D.Y Size

Example

```
INFO, /Structure, !D
SET SCREEN, !D.X Size/10., !D.X Size/2.
SET SCREEN, !D.X Size/10., !D.X Size/2.1, $
   !D.Y Size/1.7, !D.Y_Size/1.1
PLOT, [3, 4, 5], Title='Upper left plot'
SET SCREEN, !D.X Size/10., !D.X_Size/2.1, $
   !D.Y Size/10., !D.Y Size/2.3
PLOT, [5, 1, 6], Title='Lower left plot', $
   /Noerase
SET SCREEN, !D.X Size/1.8, !D.X Size/1.1, $
   !D.Y_Size/1.7, !D.Y_Size/1.1
PLOT, [3, 4, 5], Title='Upper right plot', $
   /Noerase
SET SCREEN, !D X Size/1.8, !D X Size/1.1, $
   !D Y_Size/10., !D_Y_Size/2.3
PLOT, [2, 4, 3], title='Lower right plot', $
   /Noerase
SET SCREEN, 1, 1, 1, 1, /Cursor
   Click with the mouse button on the lower left corner of the
   desired plotting area and then again for the desired upper right
   corner.
PLOT, [3, 5, 4]
SET SCREEN, !D X Size/10., !D_X_Size/1.1, $
   !D Y Size/10., !D_Y_Size/1.1
PLOT, [3, 4, 5],
   Title='Without setting the region keyword'
SET SCREEN, !D X Size/10., !D X Size/1.1, $
   !D Y Size/10., !D Y Size/1.1, Region
PLOT, [3, 4, 5],
   Title='With the region keyword set', $
   /Noerase
```

See Also

!P.Region, !P.Position, SET_VIEWPORT

SET_SHADING Procedure

Modifies the light source shading parameters affecting the output of SHADE SURF and POLYSHADE.

Usage

SET SHADING

Input Parameters

None.

Input Keywords

Gouraud — Controls the method of shading the surface polygons of the POLYSHADE procedure:

- If set to nonzero (the default), the Gouraud shading method is used.
- Otherwise, each polygon is shaded with a constant intensity.

Gouraud shading interpolates intensities from each vertex along each edge. Then, when scan converting the polygons, the shading is interpolated along each scan line from the edge intensities. Gouraud shading is slower than constant shading, but usually results in a more realistic appearance.

The SHADE_SURF procedure always uses the Gouraud method.

Light - A 3-element vector specifying the direction of the light source. The default light source vector is [0, 0, 1], with the light rays parallel to the Z axis.

Reject — If set (the default), causes polygons to be rejected as being hidden if their vertices are ordered in a clockwise direction as seen by the viewer.

• You should always set *Reject* when rendering enclosed solids whose original vertex lists are in counterclockwise order.

You may also choose to set *Reject* when rendering surfaces that are not closed or are not in counterclockwise order, although this may cause shading anomalies at boundaries between visible and hidden surfaces to occur.

Discussion

SET SHADING keywords let you control the light source direction, shading method, and the rejection of hidden surfaces. SET SHADING first resets its keywords to their default values. The values specified in the call then overwrite the default values.

Example

This example creates a spherical volume dataset and then renders two isosurfaces from that dataset. The first isosurface does not use the Gouraud method of shading but instead shades each polygon with a constant intensity. This is achieved by using SET SHADING with the Gouraud keyword set to 0. The second isosurface uses the Gouraud method of shading, which is achieved by using SET SHADING with the Gouraud keyword set to 1.

```
sphere = FLTARR(20, 20, 20)
   Create a three-dimensional single precision, floating-point array.
FOR x = 0, 19 DO FOR y = 0, 19 DO FOR $
   z = 0, 19 DO sphere(x, y, z) = $
   SQRT((x-10)^2 + (y-10)^2 + (z-10)^2)
       Create the spherical volume dataset.
SHADE_VOLUME, sphere, 7, v, p
   Find the vertices and polygons at a contour level of 7.
SURFACE, FLTARR(2, 2), /Nodata, /Save, $
   Xrange = [0, 20], Yrange = [0, 20], $
   Zrange = [0, 20], Xstyle = 4, Ystyle = 4, $
   Zstyle = 4
       Set up an appropriate three-dimensional transformation.
SET SHADING, Gouraud = 0
   Turn Gouraud shading off.
```

image = POLYSHADE(v, p, /T3d) Render the image.

TV, image
Display the image.

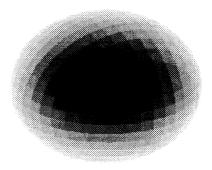


Figure 2-46 Spherical isosurface with constant intensity shading.

SET_SHADING, Gouraud = 1
Turn Gouraud shading on.

image = POLYSHADE(v, p, /T3d)
Render the image.

TV, image
Display the image.

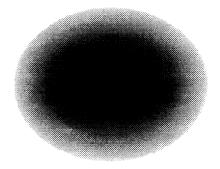


Figure 2-47 Spherical isosurface with Gouraud shading.

See Also

POLYSHADE, SHADE_SURF, SHADE_SURF_IRR

SET_SYMBOL Procedure

(VMS Only) Defines a DCL interpreter symbol for the current process.

Usage

SET_SYMBOL, name, value

Input Parameters

name – A scalar string containing the name of the symbol to be defined.

value — A scalar string containing the value with which *name* will be defined.

Input Keywords

Type – Indicates the VMS table into which name will be placed:

- 1 Specifies the local symbol table (the default).
- 2 Specifies the global symbol table.

See Also

DELETE_SYMBOL, DELLOG, GET_SYMBOL, SETLOG, TRNLOG

For more information, see the *VAX/VMS DCL Dictionary*. Information oriented to C programmers can be found in the *VMS System Services Manual*.

SETUP KEYS Procedure

Standard Library procedure that sets up the functions keys for keyboards known to PV-WAVE.

Usage

SETUP KEYS

Parameters

None.

Input Keywords

App_Keypad - Specifies the escape sequences for the group of keys in the numeric keypad, enabling these keys to be programmed within PV-WAVE.

Eightbit — Indicates that the 8-bit versions of the escape codes should be used instead of the default 7-bit versions when the VT200 function key definitions are entered. Eightbit is only valid if the VT200 keyword is also used.

HP9000 - Specifies that function key definitions for an HP-9000 Series 300 keyboard should be established.

The upper right-hand group of four keys (at the same height as the function keys) are called <BLANK1> throught <BLANK4>, since they have no written labels.

Keys defined to have labels beginning with a capital <K> belong to the numeric keypad group; for example, <K9> refers to keypad key <9>.

Mips — Specifies that function key definitions for a MIPS RS series keyboard should be established.

Num Keypad - Disables programmability of the numeric keypad.

Sun — Specifies that function key definitions for a Sun-4 keyboard should be established.

VT200 — Specifies that function key definitions for a VT200 keyboard should be established.

Discussion

SETUP_KEYS is handy for initializing or resetting the function key definitions of a particular keyboard, since the number of function keys, their names, and the escape sequences they send vary between various keyboards.

If no keyboard-specific keyword is specified, SETUP_KEYS uses the system variable !Version to determine the type of machine running PV-WAVE. It assumes the keyboard is the same type as the machine.

The function key definitions can be examined by running the SETUP_KEYS procedure and then typing INFO, /Keys.

Example

SETUP KEYS, /Eightbit, /VT200

Establishes function key definitions for a VT200 keyboard using 8-bit versions of escape codes.

See Also

DEFINE KEY, SETDEMO, !Version

SET VIEW3D Procedure

Generates a 3D view, given a view position and a view direction.

Usage

SET VIEW3D, viewpoint, viewvector, perspective, izoom, viewup, viewcenter, winx, winy, xr, yr, zr

Input Parameters

viewpoint — A three-element vector containing the point from which to view the data in data coordinates.

viewvector — A three-element vector containing the direction to look.

perspective — The perspective projection distance. The smaller the projection distance, the more "severe" the projection is. The larger the projection distance, the more "isometric" the projection is.



To prevent a perspective projection, set perspective to 0 (or less than zero).

izoom – The magnification factor for the projection.

viewup — A two-element vector containing the final 2D view up vector. For a "right-side-up" view, set this parameter to [0.0, 1.0].

viewcenter — A two-element vector containing the window location on which to place the viewpoint. It is in normal coordinates and is usually set to [0.5, 0.5].

winx, winy — The X and Y dimensions, respectively, of the plot window in device coordinates. Typically, winx and winy are set to the X and Y size of the current PV-WAVE window.

xr, yr, zr — Two-element vectors containing the minimum and maximum X, Y, and Z values, respectively, found in the data to be plotted. The minimum value is in xr(0), yr(0), and zr(0); the maximum value in xr(1), yr(1), and zr(1).

Keywords

None.

Discussion

SET_VIEW3D creates a view transformation that preserves the correct aspect ratio of the data, even if the plot window is non-square.

Note

SET_VIEW3D changes the system viewing matrix !P.T, as well as the system variables, !X.S, !Y.S, and !Z.S, which handle conversion from data coordinates to normal coordinates. (These system variables are described in Chapter 4, System Variables.

Examples

See the *Examples* section in the description of the POLY_DEV routine.

See Also

CENTER VIEW

SET VIEWPORT Procedure

Standard Library procedure that establishes a new position for the rectangular plot area based on input values specified using the normalized coordinate system.

Usage

SET VIEWPORT, xmin, xmax [, ymin, ymax]

Input Parameters

xmin — The position of the left edge of the rectangular plot area in normal coordinates.

xmax — The position of the right edge of the rectangular plot area in normal coordinates.

ymin — The position of the bottom edge of the rectangular plot area in normal coordinates.

ymax – The position of the top edge of the rectangular plot area in normal coordinates.

Input Keywords

Cursor — Lets you set the region for the rectangular plot area interactively with the mouse:

- If nonzero, lets you set the boundaries by clicking the mouse at the corners of the rectangular plot area that you want to use. Click first to set the lower-left corner, then once again to set the upper-right corner. Any input parameters for SET_VIEWPORT are ignored.
- If zero, uses the input parameters as specified.

Region — If present, uses the system variable !P.Region to set the plot area. The default is to use !P.Position.

Discussion

SET_VIEWPORT is used to set up the area of the display that will be used for plotting. It is identical to the SET_SCREEN procedure, except that the input parameters are in normalized coordinates for SET_VIEWPORT.

If only *xmin* and *xmax* are provided, the values for *ymin* and *ymax* are calculated. Calling SET_VIEWPORT with four input parameters is the same as setting the !P.Postion system variable or using the *Position* keyword with the plotting commands.

SET_VIEWPORT overrides the effect of the system variables !X.Margin and !Y.Margin.

Example

```
SET_VIEWPORT, .2, .5
PLOT, [3, 5, 2]
SET VIEWPORT, .1, .45, .55, .90
PLOT, [3, 4, 5], Title='Upper left plot'
SET VIEWPORT, .1, .45, .1, .45
PLOT, [5, 1, 6], Title='Lower left plot', $
   /Noerase
SET_VIEWPORT, .55, .90, .55, .90
PLOT, [3, 4, 5], Title='Upper right plot', $
   /Noerase
SET VIEWPORT, .55, .90, .1, .45
PLOT, [2, 4, 3], Title='Lower right plot', $
   /Noerase
SET_VIEWPORT, 1, 1, 1, 1, /Cursor
   Click with the mouse button on the lower left corner of the
   desired plotting area and then again for the desired upper right
   corner.
PLOT, [3, 5, 4]
SET VIEWPORT, .2, .8, .2, .8
PLOT, [3, 4, 5], $
   Title='Without setting the region keyword'
```

```
SET VIEWPORT, .2, .8, .2, .8, /Region
PLOT, [3, 4, 5], $
   Title='With the region keyword set', $
   /Noerase
```

See Also

!P.Position, !P.Region, SET SCREEN

SET XY Procedure

Standard Library procedure that sets the default data axis range for either the X or Y axis. It should only be used for emulating Version 1 of PV-WAVE.

Usage

SET XY, xmin, xmax [, ymin, ymax]

Input Parameters

xmin – The minimum default value for the X axis.

xmax — The maximum default value for the X axis.

ymin — The minimum default value for the Y axis.

ymax — The maximum default value for the Y axis.

Keywords

None.

Discussion

SET XY is used for setting the range of data values that will be drawn in the rectangular display window. (Normally the axis range is determined automatically by scanning the data.)

Note that you should only use SET_XY if you are trying to emulate Version 1 of PV-WAVE. With more recent versions of PV-WAVE, you should use the system variables !X.Range and !Y.Range, or their corresponding keywords.

Caution

SET_XY sets the Range, Crange, and S (scaling) fields of the system variables !X and !Y. This may cause subsequent plots of otherwise familiar data to appear skewed.

Example

```
PLOT, [12, 26, 35, 44], [50, 43, 89, 70]

PLOT, [12, 26, 35, 44], [50, 43, 89, 70], $

XRange=[10, 50], YRange=[40, 100]

SET_XY, 10, 50, 40, 100

PLOT, [12, 26, 35, 44], [50, 43, 89, 70]
```

See Also

AXIS, PLOT, SET_VIEWPORT
!X.Range, !Y.Range, !X.Style, !Y.Style system variables
XRange, YRange, XStyle, YStyle plotting keywords

For more information, see *Drawing X Versus Y Plots* on page 61 of the *PV*-WAVE User's Guide.

SHADE SURF Procedure

Creates a shaded surface representation of a regular or nearly regular gridded surface, with shading from either a light source model or from a specified array of intensities.

Usage

SHADE_SURF, z [, x, y]

Input Parameters

z - A two-dimensional array containing the values that make up the surface. If x and y are supplied, the surface is plotted as a function of the X,Y locations specified by their contents. Otherwise, the surface is generated as a function of the array index of each element of z.

x - A vector or two-dimensional array specifying the X coordinates for the contour surface. If x is a vector, each element of xspecifies the X coordinate for a column of z.

For example, x(0) specifies the X coordinate for z(0, *). If x is a two-dimensional array, each element of x specifies the X coordinate of the corresponding point in $z(x_{ij})$ specifies the X coordinate for z_{ii}).

y - A vector or two-dimensional array specifying the Y coordinates for the contour surface. If a vector, each element of y specifies the Y coordinate for a row of z.

For example, y(0) specifies the Y coordinate for z (*, 0). If y is a two-dimensional array, each element of y specifies the Y coordinate of the corresponding point in $z(y_{ij})$ specifies the Y coordinate for z_{ii}).

Input Keywords

Shades — An array expression, of the same dimensions as z, containing the color index at each point. The shading of each pixel is interpolated from the surrounding **Shades** values. For most displays, this parameter should be scaled into the range of bytes. If this keyword is omitted, light source shading is used.

Other keywords are listed below. For a description of each keyword, see Chapter 3, *Graphics and Plotting Keywords*.

Ax	Position	XTicklen	YTickv
Az	Save	XTickname	YTitle
Background	Subtitle	XTicks	
Channel	T3d	XTickv	ZCharsize
Charsize	Thick	XTitle	ZGridstyle
Charthick	Tickformat		ZMargin
Clip	Ticklen	YCharsize	ZMinor
Color	Title	YGridstyle	ZRange
Data		YMargin	ZStyle
Device	XCharsize	YMinor	ZTickformat
Font	XGridstyle	YRange	ZTicklen
Gridstyle	XMargin	YStyle	ZTickname
Noclip	XMinor	YTickformat	ZTicks
Nodata	XRange	YTicklen	ZTickv
Noerase	XStyle	YTickname	ZTitle
Normal	XTickformat	YTicks	ZValue

Output Keywords

Image — The name of a variable into which the image containing the shaded surface is stored. If this keyword is omitted, the image is displayed but not saved.

Discussion

SHADE_SURF is similar to the SURFACE procedure. Given a regular or near-regular grid of elevations, SHADE_SURF produces a shaded surface representation of the data with the hidden surfaces removed.

If the graphics output device has scalable pixels, the output image is scaled so that its largest dimension is less than or equal to 512. When outputting to a device that prints black on a white background (e.g., PostScript devices), pixels containing the background color index of 0 are set to white.

Use the SET SHADING procedure to control the direction of the light source and other shading parameters.

Note /

If the T3D keyword is set, the 3D to 2D transformation matrix contained in !P.T must project the Z axis to a line parallel to the device Y axis, or errors will occur.

If the X,Y grid is not regular or nearly regular, errors in hidden line removal will likely occur. In this case, you should use the SHADE_SURF_IRR procedure.

Caution |

SHADE SURF is not supported on Tektronix terminals or the 4510 Rasterizer. If you try to display shaded image on such a device, PV-WAVE may abort. This is because of a limitation in the range of image coordinates available on Tektronix devices.

SHADE SURF is also unsupported on VT240 terminals.

Example 1

This example uses SHADE SURF to display a shaded surface representation of the function

$$f(x,y) = x\sin(y) + y\cos(x) - \sin(\frac{xy}{4})$$

where

$$(x, y) \in \{ \mathbb{R}^2 | x, y \in [-10, 10] \}$$

x = FINDGEN(101)/5 -10

Create 101-element vector of x-coordinates such that $x \in [-10, 10]$.

y = x

Make the vector of y-coordinates the same as the vector of x-coordinates.

z = FLTARR(101, 101)

Create a 101-by-101 array to hold the function values.

FOR i = 0, 100 DO BEGIN &\$ z (i, *) = x(i)*SIN(y) + y*COS(x(i)) - \$ SIN(0.25*x(i)*y)

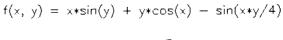
Insert the function values into z. Note that z is filled columnwise instead of elementwise.

SHADE_SURF, z, x, y, Ax = 50, XCharsize = 2, \$
YCharsize = 2, ZCharsize = 2

Display the shaded surface. The Ax keyword is used to specify the desired angle of rotation about the x-axis. The XCharsize, YCharsize, and ZCharsize keywords are used to enlarge the characters used to annotate the axes.

XYOUTS, 118, 463, \$
" $f(x, y) = x*\sin(y) + y*\cos(x) - \sin(x*y)/4$)",\$
Charsize = 2, /Device

Place a title in the window. Note that the CURSOR procedure with the Device keyword was used to locate the proper position for the title.



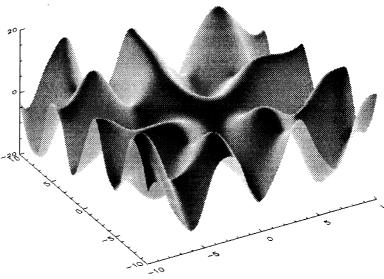


Figure 2-48 Shaded surface with title.

Example 2

Users often wish to store the data that describes a surface in a file. Each row of data in the file is a point on the surface so there are three columns of data in the file. The first column contains x-coordinates of the points, the second column contains y-coordinates of the points, and the third column contains z-coordinates of the points.

This example creates data describing the surface defined by the function

$$f(x, y) = xy\cos(0.575xy) - 10(x^2 + y^2)$$

where

$$(x, y) \in \{ \mathbb{R}^2 | x, y \in [-10, 10] \}$$

and places that data in the file shsurf.dat in the columnar format described above. The data is then read from the file shsurf.dat, placed in the data structures expected by SHADE_SURF, and displayed.

- .RUN
- FUNCTION f, x, y
- RETURN, x * y * cos(0.575 * x * y) 10 * \$
- $-(x^2 + y^2)$
- END

Define the function.

$$x = FINDGEN(101)/5 - 10$$

y = x

Create vectors containing the x- and y- coordinates of the region of the xy-plane over the surface to be viewed.

z = FLTARR(101, 101)

Create a z-dimensional array large enough to hold the function values on the surface.

OPENW, 1, 'shsurf.dat'

Open the file that is to hold the data describing the surface.

ENDFOR

Write the x-, y-, and z-coordinates of points on the surface to the file shsurf.dat.

FLUSH, 1

Flush any buffered output to the output file and close the file.

CLOSE, 1

q = FLTARR(3, 101 * 101)

Read the data describing the surface from shsurf.dat and display the surface. First, create an array large enough to hold the data contained in shsurf.dat.

OPENR, 1, 'shsurf.dat' Open shsurf.dat for reading.

RMF, 1, q, 101 * 101, 3

Use RMF to read the data from the file. This reads the data as if it were a matrix.

CLOSE, 1

Close the input file.

q = TRANSPOSE(q)

Convert the matrix in Q to an array.

nx = q(0, *)

Copy the x-coordinates into nx.

nx = TRANSPOSE(REFORM(nx, 101, 101))Convert nx from a vector to a two-dimensional array.

ny = q(1, *)Copy the y-coordinates into ny.

ny = TRANSPOSE(REFORM(ny, 101, 101))Convert ny from a vector to a two-dimensional array.

nz = q(z, *)

Copy the z-coordinates into nz.

nz = Transpose(REFORM(nz, 101, 101))Convert nz from a vector to a z-dimensional array.

SHADE SURF, nz, nx, ny, Ax = 50, \$

XCharsize = 2, YCharsize = 2, ZCharsize = 2 Display the surface. The Ax keyword is used to specify the desired angle of rotation about the x-axis. The XCharsize, YCharsize, and ZCharsize keywords are used to enlarge the characters used to annotate the axes.

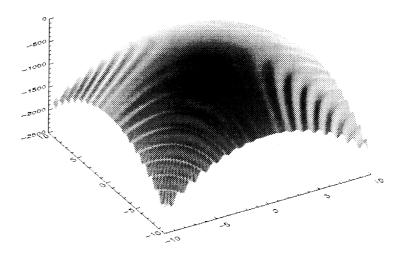


Figure 2-49 Shaded surface defined by $f(x, y) = xy \cos(0.575xy) - 10(x^2 + y^2)$.

Example 3

SHADE_SURF, DIST(100), Ax=60

See Also

SHADE_SURF_IRR, SET_SHADING, SURFACE

SHADE_SURF_IRR Procedure

Standard Library routine that creates a shaded-surface representation of a semiregularly gridded surface, with shading from either a light source model or from a specified array of intensities.

Usage

SHADE_SURF_IRR, z [, x, y]

Input Parameters

- z A two-dimensional array containing the values that make up the surface. If x and y are supplied, the surface is plotted as a function of the X,Y locations specified by their contents. Otherwise, the surface is generated as a function of the array index of each element of z.
- x A two-dimensional array specifying the X coordinates for the contour surface. Each element of x specifies the X coordinate of the corresponding point in z (x_{ij} specifies the X coordinate for z_{ij}).
- y A two-dimensional array specifying the Y coordinates for each elevation. Each element of y specifies the Y coordinate of the corresponding point in z (y_{ij} specifies the Y coordinate for z_{ii}).

Input Keywords

Shades — An array expression, of the same dimensions as z, containing the color index at each point. The shading of each pixel is interpolated from the surrounding **Shades** values. For most displays, this parameter should be scaled into the range of bytes. If this keyword is omitted, light source shading is used.

Other keywords are listed below. For a description of each keyword, see Chapter 3, Graphics and Plotting Keywords.

Ax	Normal	XTicks	YTitle
Az	Position	XTickv	
Background	Save	XTitle	ZCharsize
Charsize	T3D		ZMargin
Clip	Ticklen	YCharsize	ZMinor
Color		YMargin	ZRange
Data	XCharsize	YMinor	ZStyle
Device	XMargin	YRange	ZTickname
Font	XMinor	YStyle	ZTicks
Noclip	XRange	YTickname	ZTickv
Nodata	XStyle	YTicks	ZTitle
Noerase	XTickname	YTickv	ZValue

Output Keywords

Image – The name of a variable into which the image containing the shaded surface is stored. If this keyword is omitted, the image is displayed but not saved.

Discussion

The input data for SHADE_SURF_IRR must be able to be represented as an array of quadrilaterals. This procedure should be used when the (x, y, z) arrays are too irregular to be drawn by the SHADE_SURF procedure, but are still semiregular.

SHADE SURF IRR is similar to the SURFACE procedure. Given a semiregular grid of elevations, it produces a shaded surface representation of the data with hidden surfaces removed.

If the graphics output device has scalable pixels, the output image is scaled so that its largest dimension is less than or equal to 512. When outputting to a device that prints black on a white background (e.g., PostScript devices), pixels containing the background color index of 0 are set to white.

Use the SET_SHADING procedure to control the direction of the light source and other shading parameters.



If the *T3D* keyword is set, the 3D to 2D transformation matrix contained in !P.T must project the Z axis to a line parallel to the device Y axis, or errors will occur.

Example

This example uses SHADE_SURF_IRR to display a shaded surface over an irregular grid. The function defining the surface is

$$f(x, y) = x \sin(y) + y \cos(x)$$

where

$$(x, y) \in {\mathbb{R}^2 | x \in [-10, 10], y \in [-5, 5]}$$

x = FLTARR(200, 100)

Create a two-dimensional array for the x-coordinates.

Create a vector to hold perturbations of the x component of a regular grid.

Compute the x-components of the irregular grid by perturbing the x-component of a regular grid.

```
y = FLTARR(200, 100)
```

Create a two-dimensional array for the y-coordinates.

```
pert = FLTARR(100)
```

Create a vector to hold perturbation of the y-component of a regular grid.

Compute the y-components of the irregular grid by perturbing the y-components of a regular grid.

.RUN

- FUNCTION f, x, y
- RETURN, x * SIN(y) + y * COS(x)
- END

Define a function of two variables to compute elevations.

$$z = FLTARR(200, 100)$$

Create a two-dimensional array for elevations.

FOR
$$i = 0$$
, 199 DO BEGIN & \$

FOR $j = 0$, 99 DO BEGIN & \$

 $z(i, j) = f(x(i, j), y(i, j)) & $$

ENDFOR & \$

ENDFOR

Compute elevations.

SHADE_SURF_IRR,
$$z$$
, x , y , $Ax = 70$

Display the shaded surface. The Ax keyword is used to specify the angle of rotation about the x-axis.

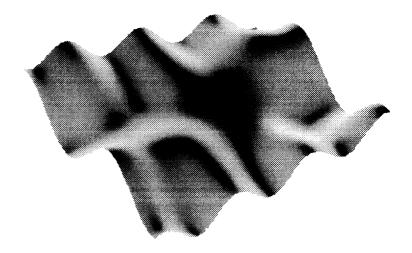


Figure 2-50 Shaded surface defined by $f(x, y) = x \sin(y) + y \cos(x)$ over irregular grid.

See Also

SHADE_SURF, SET_SHADING, SURFACE

SHADE_VOLUME Procedure

Given a 3D volume and a contour value, produces a list of vertices and polygons describing the contour surface.

Usage

SHADE VOLUME, volume, value, vertex, poly

Input Parameters

volume — An array with three dimensions containing the dataset to be contoured. If the volume array is dimensioned (D_0, D_1, D_2) , the resulting vertices range as follows:

- In X, they range between 0 and $D_0 1$.
- In Y, they range between 0 and $D_1 1$.
- In Z, they range between 0 and $D_2 1$.

value — A scalar containing the contour value.

Output Parameters

vertex — The name of a variable to receive the vertex array. This variable will be set to a (3, n) floating-point array, suitable for input to the MESH or POLYSHADE procedure.

poly – The name of a variable to receive the polygon list, an m-element longword array. This list describes the vertices of each polygon and is suitable for input to MESH or POLYSHADE.

Input Keywords

Low – If present and nonzero, indicates that the low side of the contour surface is to be displayed and that the contour surfaces enclose high data values. Otherwise, it is assumed that the high side of the contour surface is to be displayed and that the contour encloses low data values. If this parameter is incorrectly specified, errors in shading will result.

Shades — An array with the same dimensions as volume. On input, it contains the user-specified shading color index for each volume element (voxel), and is converted to byte type before use.

Output Keywords

Shades — On output, this input array is replaced by another array containing the shading value for each vertex, contained in vertex.

Discussion

SHADE_VOLUME computes the polygons that describe a threedimensional contour surface. Each voxel is visited to find the polygons formed by the intersections of the contour surface and the voxel edges.

You can obtain shading from either a single light-source model or from the values you specify with Shades.

The surface produced by SHADE VOLUME may then be displayed as a shaded surface with the POLYSHADE procedure.

This routine is limited to processing datasets that will fit in memory.

Example

The following procedure shades a volume passed as a parameter. It uses SURFACE to establish the viewing transformation. It then calls SHADE VOLUME to produce the vertex and polygon lists, and POLYSHADE to draw the contour surface.



For another example demonstrating SHADE VOLUME, see the POLYSHADE function.

PRO ShowVolume, vol, thresh, Low=low Display the contour surface of a volume.

s = SIZE(vol)Get the dimensions.

```
IF s(0) NE 3 THEN PRINT, 'Error' Flag an error if s is not a 3D array.
```

Use SURFACE to establish 3D transformation and coordinate ranges.

IF N_ELEMENTS (low) EQ 0 THEN low = 0

Make the default be to view the high side of the contour surface.

SHADE_VOLUME, vol, thresh, v, p, Low=low Produce the vertices and polygons.

TV, POLYSHADE(v, p, /T3D)

Produce the image of the surface and display.

END

See Also

POLYSHADE, SURFACE

For information on volume visualization, see Chapter 6, Advanced Rendering Techniques, in the PV-WAVE User's Guide.

The method used by SHADE_VOLUME is that described by Klemp, McIrvin and Boyd in "PolyPaint — A Three-Dimensional Rendering Package," *American Meteorology Society Proceedings*, Sixth International Conference on Interactive Information and Processing Systems, 1990.

This method is similar to the marching cubes algorithm described by Lorenson and Cline in "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, pp. 163-169, 1987.

SHIFT Function

Shifts the elements of a vector or array along any dimension by any number of elements.

Usage

```
result = SHIFT(array, shift_1, ..., shift_n)
```

Input Parameters

array — The array to be shifted.

 $shift_i$ — The shift parameters (see Discussion below).

Returned Value

result — The shifted array.

Keywords

None.

Discussion

The SHIFT function is used to perform a circular shift upon the elements of a vector or an array. The resulting array has the same dimension and data type as the input array. Shifts are handled similarly, regardless of the number of dimensions in the input array, as detailed below:

- Shifts on One-Dimensional Arrays A shift performed on a one-dimensional array (a vector) shifts the contents of each element to the right or left, depending on the number of elements specified in the second parameter: a positive number shifts elements to the right, while a negative number shifts them to the left.
- Shifts on Two-Dimensional Arrays A shift performed on a two-dimensional array (such as a raster image) is done in a similar way. The contents of entire rows and/or columns are

shifted to the rows above or below, or to the columns to the right or left, depending on the number of rows and columns specified by the second and third parameters of the process, respectively.

Positive numbers for the second and third parameters shift rows in an up direction (or columns to the right), while negative numbers shift rows in a down direction (or columns to the left).

• Shifts on Arrays with More than Two Dimensions — For arrays of more than two dimensions, the parameter $shift_n$ specifies the shift applied to the nth dimension. For example, $shift_1$ specifies the shift along the first dimension. If you specify 0 for $shift_n$, this means that no shift is to be performed along that dimension.

Regardless of the number of dimensions, all shifts are circular, meaning that values that are pushed off the beginning or end of the array by the shift operation are automatically inserted back onto the opposite end of the array. No values in the array are lost.

If only one shift parameter is present and the parameter is an array, the array is treated as a vector.

Sample Usage

Typical uses of the SHIFT function include:

- To force the elements of one array to align with the elements of another array.
- To force the elements of one array to be misaligned with the elements of another array (some statistical analysis techniques require this).
- To line up (or register) the edges of an image to match those
 of another image. This can be used to compensate for an
 image that was initially digitized out of alignment with respect
 to the edges of another image.
- To shift the beginning and ending point of a color table in an image.

To do an edge enhancement technique for images commonly known as Shift and Difference Edge Enhancement (see Example 2 below for more information).

Example 1 — One-dimensional Shifting

The following demonstrates using SHIFT with a vector:

```
a = INDGEN(5)
   Make a small vector.
```

Print the vector, the vector shifted one position to the right, and the vector shifted one position to the left.

Executing these statements gives the result:

0	1	2	3	4
4	0	1	2	3
1	2	3	4	0

Notice how elements of the vector that shift off the end wrap around to the other end. This "wrap around" occurs when shifting arrays of any dimension.

Example 2 — Shift and Difference Edge Enhancement

SHIFT can be used to create an edge enhancement technique for images. In this technique, a copy of an array may be shifted by one or more elements (pixels) in any direction, and then subtracted from the original image.

When performed on a binary image (containing only black and white pixels), edges that are opposite the direction of the shift are enhanced.

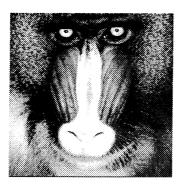
When performed on a gray scale or pseudocolor image, an embossing effect is created opposite the direction of the shift.

By carefully selecting the direction and amount of the shift, you can make certain details (for example, only vertical or horizontal lines) be discernible in an otherwise jumbled image.

Typically, single-element (one pixel) shifts are most pronounced, while any shift beyond ten elements (pixels) tends to start blurring the features in the image.

For example, to shift a mandril image to highlight the edges to the left of each feature, you would first run the SHIFT function and then subtract the resulting image from the original image to create a third image:

```
shift_mandril = SHIFT(mandril, 1, 0)
shift_diff_mand = mandril - shift mandril
```



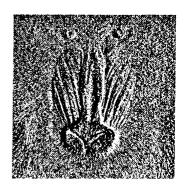
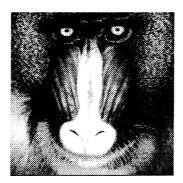


Figure 2-51 The SHIFT function described above has been used with this 512-by-512 mandril image for edge enhancement. Notice how the dark edges to the left of each feature in the image are highlighted by using SHIFT with a positive number for the first parameter (which causes the image to be shifted to the right).

To shift a mandril image to highlight the edges below and to the left of each feature:

```
shift_mandril = SHIFT(mandril, 1, 1)
shift_diff_mand = mandril - shift_mandril
```



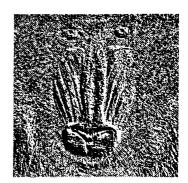


Figure 2-52 Notice how the dark edges to the southwest of each feature in the image are now highlighted by using the SHIFT function with a positive number for the first and second parameter (which causes the image to be shifted both up and to the right).

See Also

CONVOL, ISHFT, ROBERTS, SOBEL

For more information about Shift and Difference Edge Enhancement, see Digital Image Processing, by Gregory Baxes, Cascade Press, Denver, 1988.

SHOW3 Procedure

Standard Library procedure that displays a two-dimensional array as a combination contour, surface, and image plot. The resulting display shows a surface with an image underneath and a contour overhead.

Usage

SHOW3, array

Input Parameters

array — The two-dimensional array to display.

Input Keywords

Ax — The angle of rotation, in degrees, about the X axis.

Az — The angle of rotation, in degrees, about the Z axis.

 C_Colors — A vector of color indices whose elements indicate which color to use in drawing the corresponding contour level.

Interp — If present and nonzero, specifies that bilinear interpolation is to be used for the pixel display. Otherwise, the nearest neighbor interpolation method is used.

Sscale — The reduction scale for the surface. If set to anything other than 1 (the default value), the image size is reduced by the specified factor. If the image dimensions are not an integral multiple of Sscale, the image is reduced to the next smaller multiple.

Bot_Image — The name of the image array to be used as the underneath image.

Discussion

You can modify SHOW3, if necessary, to customize the contour and surface commands to your satisfaction (e.g., use different colors, skirts, line styles, and contour levels). See the descriptions of the CONTOUR and SURFACE routines for details.



When you are displaying larger images (say 50-by-50), the display produced by SHOW3 can become too "busy." If this happens, try using the SMOOTH and/or REBIN procedures to smooth the surface plot.

Caution

The SHOW3 procedure is not supported on Tektronix terminals or the 4510 Rasterizer. If you try to display a SHOW3 image on such a device, PV-WAVE may abort. This is because of a limitation in the range of image coordinates available on Tektronix devices.

Example

This example uses the Pike's Peak elevation and snowpack images found in the PV-WAVE directory wave/data to recreate the display that appears on the cover of this manual:

```
OPENR, 1, !data_dir + 'pikeselev.dat'
    Open the file to read.
pikes = FLTARR(60, 40)
    Create the data array for the pikes elevation data.
READF, 1, pikes
    Read in the formatted file.
CLOSE, 1
   Close the file.
OPENR, 2, !data_dir + 'snowpack.dat'
   Open the file to read.
snow = FLTARR(60, 40)
   Create the data array for snow pack data.
READF, 2, snow
   Read in the formatted file.
CLOSE, 2
   Close the file.
LOADCT, 5
   Load color table number 5.
```

SHOW3, SMOOTH(pikes, 3), Bot_image=snow, \$ /Interp

Produce the combined display.

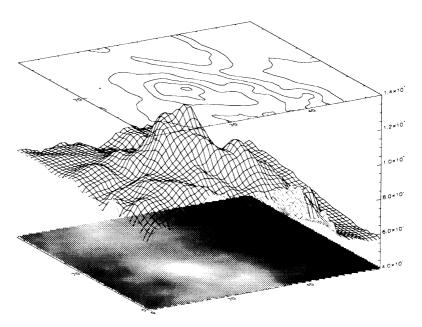


Figure 2-53 Combination image, surface, and contour produced with SHOW3.

See Also

CONTOUR, REBIN, SURFACE, SMOOTH

For information on interpolation methods, see *Efficiency and Accuracy of Interpolation* on page 170 of the *PV*-WAVE User's Guide.

SIGMA Function

Standard Library function that calculates the standard deviation value of an array. (Optionally, it can also calculate the standard deviation over one dimension of an array as a function of the other dimensions.)

Usage

result = SIGMA(array [, npar, dim])

Input Parameters

array — The array to be processed. Can be of any data type except

npar — The number of parameters (the default value is 0).

dim — The dimension over which to calculate the standard deviation.

Returned Value

result — The standard deviation of array, or the standard deviation over one dimension of array as a function of dim.

Keywords

None.

Discussion

The SIGMA function is useful in a variety of data and statistical analyses for estimating thresholds and evaluating the significance of variance.

The number of degrees of freedom in SIGMA is equal to the number of elements in array minus the value supplied in the optional npar. In other words:

degrees of freedom = #_of elements_in_array - npar

The degrees of freedom affects the value of the standard deviation. Specifically, the value of the standard deviation varies as one over the square root of the number of degrees of freedom.

If dim is used, then the result of SIGMA is an array with the same dimensions as the input array, except for the dimension specified. Each element in the resulting array is the standard deviation of the corresponding vector in the input array.

The dimension specified in a SIGMA call must be valid for the array passed; otherwise, the input array is returned as the output array.

If array is an array with dimensions of (3,4,5), then the command:

```
is equivalent to the commands:

std = FLTARR(3,5)
```

std = SIGMA(array, 2, 1)

```
FOR j = 0, 4 DO BEGIN
FOR i = 0, 2 DO BEGIN
STD(i, j) = SIGMA(array(i, *, j), 2)
ENDFOR
ENDFOR
```

Example

PRINT, SIGMA(a, 1, 0)

0.00000

0.00000

0.00000

Print the standard deviation of the column elements.

PRINT, SIGMA(a, 1, 1)

2.00000

2.00000

2.00000

Print the standard deviation of the row elements.

See Also

STDEV

SIN Function

Returns the sine of the input variable.

Usage

result = SIN(x)

Input Parameters

x — The angle, in radians, that is evaluated.

Returned Value

result — The trigonometric sine of x.

Keywords

None.

Discussion

If x is of double-precision floating-point or complex data type, SIN yields a result of the same type. All other types yield a singleprecision floating-point result.

SIN handles complex numbers in the following way:

$$sin(x) = complex((sin(r)cosh(i), cos(r)sinh(i))$$

where r and i are the real and imaginary parts of x.

If x is an array, the result of SIN has the same data type, with each element containing the sine of the corresponding element of x.

Example

The following commands produce a dampened sine wave.

```
x = FINDGEN(200)
PLOT, 10000 * SIN(x/5) / EXP(x/100)
```

See Also

COS, COSH, SINH

For a list of other transcendental functions, see *Transcendental Mathematical Functions* on page 20, in Volume 1 of this reference.

SINDGEN Function

Returns a string array with the specified dimensions.

Usage

$$result = SINDGEN(dim_1, ..., dim_n)$$

Input Parameters

 dim_i — The dimensions of the result. May be any scalar expression. Up to eight dimensions may be specified.

Returned Value

result — An initialized string array. If the resulting array is treated as a one-dimensional array, then its initialization is given by the following:

array (i) = STRING (i), for
$$i = 0, 1, ..., \left(\prod_{j=1}^{n} D_j - 1 \right)$$

Keywords

None.

Discussion

Each element of the array is set to the string representation of the value of its one-dimensional subscript, using the default formatting rules of PV-WAVE. These rules are described in Table 8-7 on page 164 of the PV-WAVE Programmer's Guide.

Example

This example creates a 4-by-2 string array.

See Also

BINDGEN, CINDGEN, DINDGEN, FINDGEN, INDGEN, LINDGEN

SINH Function

Returns the hyperbolic sine of the input variable.

Usage

$$result = SINH(x)$$

Input Parameters

x — The angle, in radians, that is evaluated.

Returned Value

result — The hyperbolic sine of x.

Keywords

None.

Discussion

SINH is defined by:

$$sinh(x) = (e^{x} - e^{-x})/2$$

If x is of double-precision floating-point or of complex data type, SINH yields a result of the same type. All other data types yield a single-precision floating-point result.

If x is an array, the result of SINH has the same dimensions, with each element containing the hyperbolic sine of the corresponding element of x.

Example

See Also

COS, COSH, SIN, TANH

For a list of other transcendental functions, see Transcendental Mathematical Functions on page 20, in Volume 1 of this reference.

SIZE Function

Returns a vector containing size and type information for the given expression.

Usage

result = SIZE(expr)

Input Parameters

expr — The expression to be evaluated.

Returned Value

result - A vector containing size and type information for expr.

Keywords

None.

Discussion

The returned vector is of longword type and has the following form:

$$[D, S_1, S_2, ..., S_D, T, E]$$

where D is the number of dimensions of EXPRESSION (zero if EXPRESSION is scalar or undefined); S_i is the size of dimension i (not present if EXPRESSION is scalar or undefined); T is the type code, encoded as shown in the table below; and E is the number of elements in EXPRESSION.

Type Code	Data Type
0	Undefined
1	Byte
2	Integer
3	Longword integer
4	Floating point
5	Double precision floating
6	Complex floating
7	String
8	Structure

See Also

N_TAGS, N_PARAMS, TAG_NAMES, N_ELEMENTS

SKIPF Procedure

(VMS Only) Skips records or files on the designated magnetic tape unit.

Usage

SKIPF, unit, files SKIPF, unit, records, r

Input Parameters

unit — A number between 0 and 9 specifying the magnetic tape unit to rewind. (Do not confuse this parameter with file logical unit numbers.)

files — The number of files to be skipped. If this number is positive, skipping is in the forward direction. Otherwise, files are skipped backwards.

records — The number of records to be skipped. If this number is positive, skipping is in the forward direction. Otherwise, records are skipped backwards.

r — If this third parameter is present, records are skipped; otherwise, files are skipped. (The value of r is never examined; its presence serves only to indicate that records are to be skipped.)

Keywords

None.

Discussion

The number of files or records actually skipped is stored in the system variable !Err. Note that when skipping records, the operation terminates immediately when the end of a file is encountered.

See Also

!Err, TAPRD, TAPWRT

For more information and sample usage, see Accessing Magnetic Tape on page 229 of the PV-WAVE Programmer's Guide.

SLICE VOL Function

Returns a 2D array containing a slice from a 3D volumetric array.

Usage

result = SLICE VOL(volume, dim, cut plane)

Input Parameters

volume - The 3D volume of data to slice.



For better results, first use VOL_PAD to preprocess the volume.

dim - The X and Y dimensions of the slice to return. Larger values for dim increase the slice resolution and execution time. dim is typically the largest of the three dimensions of volume.

cut plane -A(3, 2) array defining the slicing plane. The elements in this (3, 2) array are interpreted as follows:

cut_plane(0, 0)	The plane's angle of rotation about the X axis.
cut_plane(1, 0)	The plane's angle of rotation about the Y axis.
cut_plane(2, 0)	Ignored.
cut_plane(0, 1)	The X coordinate of the center of the plane.
cut_plane(1, 1)	The Y coordinate of the center of the plane.
cut_plane(2, 1)	The Z coordinate of the center of the plane.



The slicing plane is rotated first about the Y axis, and then about the X axis.

Returned Value

result — A 2D array containing a slice from a 3D volumetric array.

Input Keywords

Degrees — If present and nonzero, *cut_plane(0:1,0)* is assumed to be in degrees.

Discussion

SLICE_VOL extracts a planar oblique slice from a volumetric (3D) array. The slicing plane is defined by the center point (X, Y, and Z), and the rotations about the Y and X axes. You can interactively define the slicing plane by calling the VIEWER procedure.

For best results, process volumes with VOL_PAD before slicing them with SLICE VOL.

Examples

For demonstrations of the SLICE_VOL procedure, use the Medical Imaging and CFD/Aerospace buttons on the PV-WAVE Demonstration Gallery.

To run the Gallery, enter wave gallery at the WAVE> prompt.

See Also

VIEWER, VOL PAD

SMOOTH Function

Smooths an array with a boxcar average of a specified width.

Usage

result = SMOOTH(array, width)

Input Parameters

array - The array to be smoothed. Must have either one or two dimensions; if it has two dimensions, the algorithm is applied over each dimension.

width — The width of the smoothing window. Should be an odd number, smaller than the smallest dimension. (If width is even, width + 1 is used instead.)

Returned Value

result - A copy of array smoothed with a boxcar average of the specified width. It has the same type and dimensions as array.

Keywords

None.

Discussion

SMOOTH is used to selectively average the elements in an array within a moving window of a given width. The result generally smooths out spikes or rapid transitions in the data, and fills in valleys and dips. This is usually called a boxcar average.

The window is a one- or two-dimensional region that traverses the input array, element by element, until it reaches the end. As the window moves across the array, all values within the window are averaged. The average value is then placed at the center of the window in the output array, while the original array is kept intact.

In a one-dimensional array, the smoothing window moves from right to left; in a two-dimensional array, it moves from upper left to lower right.

Note

The window may be any size as long as it is smaller than the array itself. One-dimensional windows of 3, 5, 7, and 9 elements are common; similarly, two-dimensional windows of 9, 25, 49, and 81 elements are also typical. The value of *width* does not appreciably affect the running time of SMOOTH.

The algorithm used by SMOOTH is shown below, where n is the number of elements in A and w is the width of the smoothing window.

When $w/2 \le i < n - w$, then:

$$R_i = (1/w) \sum_{j=0}^{w-1} A_{i+j-w/2}$$

Otherwise, $R_i = A_i$.

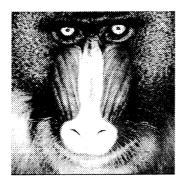
Sample Usage

Typical uses for the SMOOTH function include:

- To remove ripples, spikes, or high frequency noise from a signal or image.
- To blur an image or set of data such that only the general trends in the data can be seen (and are thereby highlighted).
- To isolate the lower spatial frequency components in an image. By subtracting a blurred image from the original image, only the higher spatial frequency components are left. (This is sometimes referred to as unsharp masking.)
- To soften sharp transitions from one color to another in a color table. This helps reduce the banding or contouring artifact evident in color tables with rapid color changes.

Example

Here is what a mandril image looks like before and after applying the SMOOTH function. For this example, a value of 7 was used for the width parameter.



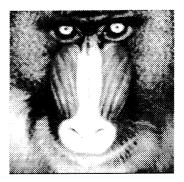


Figure 2-54 The SMOOTH function has been used to soften the sharp contrasts in this 512-by-512 mandril image.

See Also

CONVOL, MEDIAN, ROBERTS, SOBEL

For more information, see Image Smoothing on page 158 of the PV-WAVE User's Guide.

SOBEL Function

Performs a Sobel edge enhancement of an image.

Usage

result = SOBEL(image)

Input Parameters

image — The two-dimensional array containing the image to be enhanced.

Returned Value

result — A two-dimensional array of integer type containing the image that has been edge-enhanced. It has the same dimensions as *image*.

Keywords

None.

Discussion

SOBEL implements an approximation of the 3-by-3 nonlinear edge enhancement operator:

$$G(j,k) = |X| + |Y|$$

where

$$X = (A_2 + 2A_3 + A_4) - (A_0 + 2A_7 + A_6)$$

$$Y = (A_0 + 2A_1 + A_2) - (A_6 + 2A_5 + A_4)$$

and where the pixels surrounding the neighborhood of the pixel F(j, k) are numbered as follows:

$$\begin{bmatrix} A_0 & A_1 & A_2 \\ A_7 & F(j,k) & A_3 \\ A_6 & A_5 & A_4 \end{bmatrix}$$

The above algorithm is a fast approximation of the SOBEL function, which is actually defined as:

$$G(j,k) = \sqrt{X^2 + Y^2}$$

Caution |

Because the result image is saved in integer format, large original data values will cause overflow. Overflow occurs when the absolute value of the result is larger than 32,767.

Sample Usage

SOBEL is commonly used to obtain an image that contains only the edges (rapid transitions between light and dark, or from one color to another) that were present in the original image. SOBEL can help enhance features and transitions between areas in an image (for example, a machine part photographed against a white background).

With this information, it is possible to identify and compare features or items in an image with those in another image, usually for verification or detection purposes. SOBEL and other edgedetection algorithms are used extensively for image processing and preprocessing for pattern recognition.

The image returned by SOBEL contains the edges present in the original image, with the brightest edges representing a rapid transition (well-defined features), and darker edges representing smoother transitions (blurred or blended features).

An original image can also be somewhat sharpened by adding or averaging the edge-detected image with the original image.

Example

Here is what an aerial image looks like before and after applying the SOBEL function.



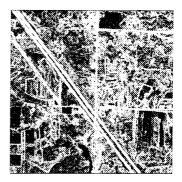


Figure 2-55 The SOBEL function has been used with this 512-by-512 aerial image to make the edges stand out with sharp contrast.

See Also

ROBERTS, CONVOL

SORT Function

Sorts the contents of an array.

Usage

result = SORT(array)

Input Parameters

array — The array to be sorted. Can be a vector or an array.

Returned Value

result - A vector of subscripts which allow access to the elements of array in ascending order.

Keywords

None.

Discussion

String arrays are sorted using the ASCII collating sequence. The result is always a vector of long integer type with the same number of elements as array.



The contents of the result are the sorted indices, not the values themselves.

Example

This example uses SORT to obtain the vector of subscripts that places the single-precision, floating-point vector a into ascending order. To obtain elements of a in sorted order, use the vector of subscripts returned by SORT as the subscript vector for a.

b = SORT(a)

Place the vector of sorting subscripts into b.

PRINT, b

3 4 1 0 2

PRINT, a(b)

Print the sorted vector by using b as the subscript vector for a.

1.00000 2.00000 3.00000 4.00000 7.00000

See Also

QUERY_TABLE, UNIQUE, WHERE, AVG, MAX, MIN, MEDIAN

SPAWN Procedure

Spawns a child process to execute a given command.

Usage

SPAWN [, command [, result]]

Input Parameters

command — The name of the command to spawn.

- If present, must be of type string. Under UNIX, command can be an array each element is passed to the child process as a separate parameter. Under VMS, command is restricted to being a scalar.
- If not present, starts an interactive shell and PV-WAVE execution suspends until the new shell process terminates. This ability is provided primarily for compatibility with the VMS version of PV-WAVE.



Shells that handle process suspension (e.g., /bin/csh) offer a more efficient way to get the same effect.

Output Parameters

result — Indicates where the result is to be directed.

- If present, places the output from the child process into a string array (one line of output per array element).
- If not present, sends the output from the child shell process to the standard output stream.

Input Keywords

F77_Unformatted - (UNIX only) If set, opens a pipe to the spawned process in a mode to do unformatted FORTRAN F77 input/output.

Noclisym - (VMS only) If set, prevents the spawned process from inheriting CLI symbols from its caller. Otherwise, the subprocess inherits all currently defined CLI symbols.



You may want to specify *Noclisym* to help prevent commands redefined by symbol assignments from affecting the spawned commands.

Nolognam – (VMS only) If set, prevents the spawned process from inheriting process logical names from its caller. Otherwise, the subprocess inherits all currently defined process logical names.



You may want to specify *Nolognam* to help prevent commands redefined by logical name assignments from affecting the spawned commands.

Noshell – (UNIX only) If present and nonzero, specifies that command should execute directly as a child process without an intervening shell process. In this case, command must be specified as a string array in which the first element is the name of the command to execute and the following parameters are the parameters to be passed to the command.

Noshell is useful when performing many spawned operations from a program and speed is a primary concern. Since no shell is present, wildcard characters are not expanded, and other tasks normally performed by the shell do not occur.

Notify — (VMS only) If set, broadcasts a message to SYS\$OUTPUT when the subprocess completes or aborts. Otherwise, no message is broadcast. This keyword should not be set unless the *Nowait* keyword is also set.

Nowait — (VMS only) If set, causes the calling process to continue executing in parallel with the subprocess. Otherwise, the calling process waits until the subprocess completes.

Sh - (UNIX only) If present and nonzero, forces the use of the Bourne shell.

Output Keywords

Count — The number of string elements returned if the output is a string variable.

Pid — A named variable for storing the process identification number of the child process.

Unit – (UNIX only) A named variable for storing the number of the file unit.

If present, causes SPAWN to create a child process in the usual manner, but instead of waiting for the specified *command* to finish, SPAWN attaches a bidirectional pipe between the child process and PV-WAVE. From the PV-WAVE session, the pipe appears as a logical file unit. The other end of the pipe is attached to the child process standard input and output.



If the *Unit* keyword is used, the *command* parameter must also be present, and *result* is not allowed.

Discussion

Under UNIX, the shell used (if any) is obtained from the SHELL environment variable. Under VMS, the DCL command language interpreter is used.

Once the child process is started, the PV-WAVE session can communicate with it using the normal input/output facilities. After the child process has done its task, the CLOSE procedure is used to delete the process and close the pipe.

Since SPAWN uses GET LUN to allocate the file unit, FREE LUN should be used to free the unit.

Example

SPAWN, 'ps -a'

See Also

CLOSE, FREE_LUN, GET_LUN, CALL_UNIX, UNIX LISTEN, UNIX_REPLY, LINKNLOAD

For background information, see Chapter 13, Interapplication Communication, in the PV-WAVE Programmer's Guide. Also see Chapter 12, Accessing the Operating System, in the PV-WAVE Programmer's Guide.

SPHERE Function

Defines a spherical object that can be used by the RENDER function.

Usage

result = SPHERE()

Parameters

None.

Returned Value

result — A structure that defines an ellipsoidal object.

Input Keywords

Color — A 256-element double-precision floating-point vector containing the color (intensity) coefficients of the object. The default is Color (*)=1.0. For more information, see the section Defining Color and Shading on page 198 of the PV-WAVE User's Guide.

Decal — A 2D array of bytes whose elements correspond to indices into the arrays of material properties. For more information, see the section *Decals* on page 201 of the *PV-WAVE User's Guide*.

Kamb - A 256-element double-precision floating-point vector containing the ambient (flat shaded) coefficients. The default is Kamb(*)=0.0. For more information, see the section Ambient Component on page 199 of the PV-WAVE User's Guide.

Kdiff — A 256-element double-precision floating-point vector containing the diffuse reflectance coefficients. The default is Kdiff(*)=1.0. For more information, see the section *Diffuse Component* on page 198 of the *PV-WAVE User's Guide*.

Ktran - A 256-element double-precision floating-point vector containing the specular transmission coefficients. The default is Ktran(*)=0.0. For more information, see the section Transmission Component on page 199 of the PV-WAVE User's Guide.

Transform — A 4-by-4 double-precision floating-point array containing the local transformation matrix whose default is the identity matrix. For more information, see the section **Setting Object and View Transformations** on page 201 of the **PV-WAVE User's Guide**.

Discussion

A SPHERE is used by the RENDER function to render ellipsoid objects, such as for spherical inverse mapping or molecular modeling (atoms). It is centered at the origin, with a radius of 0.5.

You can alter its diameter and orientation with the *Transform* keyword.

Examples

```
ds = 6
checks = BYTARR(ds, ds)
checks(*) = 255
```

```
FOR x=0, ds - 1 DO $
   FOR y=0, ds - 1 DO $
     IF ((x \mod 2) EQ (y \mod 2)) THEN $
       checks(x, y) = 128
   Create a 6-by-6 checkerboard image.
two = FLTARR(256)
two(128) = 0.5 \& two(255) = 1.0
   Set checks to be full and one-half intensity.
s = SPHERE(Decal=checks, Color=two)
TV, RENDER(s)
   Render the sphere with the checkerboard decal mapped on.
```

See Also

CONE, CYLINDER, MESH, RENDER, VOLUME

For more information, see Ray-tracing Rendering on page 196 of the PV-WAVE User's Guide.

SPLINE Function

Standard Library function that performs a cubic spline interpolation.

Usage

result = SPLINE(x, y, t [, tension])

Input Parameters

- x A vector containing the independent coordinates of the dataset. Must be monotonic and increasing.
- y A vector containing the dependent coordinates of the dataset.
- t A vector containing the independent coordinates for which the ordinates are desired. Must be monotonic and increasing.

tension — The amount of tension to be put on the spline curve. The default value is 1.0 (see Discussion below).

Returned Value

result — A vector containing interpolated ordinate values. In other words, result(i) = value of the function at T(i).

Keywords

None.

Discussion

SPLINE performs a cubic spline interpolation of the input vector in order to obtain a vector of interpolated dependent values. The dependent values are calculated at the independent values given by the vector *t*.

A tension curve always passes through each known data point. The optional *tension* parameter controls the smoothness of the fitted curve. The higher the tension, the more closely the curve

approximates the set of line segments that connect the data points. A lower tension produces a smoother curve that may deviate considerably from the straight-line path between points.

When tension is set close to zero (e.g., 0.01), the curve is virtually a cubic spline fit. When tension is set to a large value (e.g., >10.0), then the fitted curve will be similar to that obtained from a polynomial interpolation, such as POLY FIT or POLYFITW.

The SPLINE function can be useful for those applications where you need to fit the data with a smoother or stiffer curve than that obtained with an interpolating polynomial. Splines also tend to be more stable than polynomials, with less possibility of wild oscillation between the data points.

Example 1

```
x = FINDGEN(10)
y = RANDOMN(seed, 10)
   Create the data.
t = FINDGEN(100)/11.
   Create a vector containing the independent points at which the
   dependent points are calculated.
PLOT, y
   Plot the original data.
PLOT, t, SPLINE(x, y, t), Xstyle=4, Ystyle=4,$
    /Noerase, linestyle=2
        Plot using 100 independent points and a default tension of
        1.
PLOT, t, SPLINE(x, y, t, 8.), Xstyle=4, $
   Ystyle=4, /Noerase, Linestyle=3
        Plot using 100 independent points and a tension of 8.
```

Example 2

$$x = [2.0, 3.0, 4.0]$$

Y formed from a quadratic function.

$$t = FINDGEN(20) / 10. + 2$$

Twenty values from 2 to 3.90 for the interpolated points.

$$z = SPLINE(x, y, t)$$

Do the interpolation.

 $y = (x - 3)^2$

See Also

CURVEFIT, GAUSSFIT, POLY_FIT, POLYFITW, SVDFIT

SQRT Function

Calculates the square root of the input variable.

Usage

$$result = SQRT(x)$$

Input Parameters

x — The value or array of values for which the square root is desired.

Returned Value

result — The square root of x.

Keywords

None.

Discussion

SQRT handles complex numbers (defined by c = a + bi), in the following manner:

$$c^{1/2} = \left[\frac{1}{2}(r+a)\right]^{1/2} \pm i\left[\frac{1}{2}(r-a)\right]^{1/2}$$

where

$$r = \sqrt{a^2 + b^2}$$

The ambiguous sign is taken to be the same as the sign of b.

If x is of double-precision floating-point or complex data type, SQRT yields a result of the same type. All other types yield a single-precision floating-point result.

If x is an array, the result of SQRT has the same dimensions, with each element containing the square root of the corresponding element of x.

Example

See Also

For more information, see ^ (the PV-WAVE symbol for power). For a list of other transcendental functions, see Transcendental Mathematical Functions on page 20, in Volume 1 of this reference.

STDEV Function

Standard Library function that computes the standard deviation and (optionally) the mean of the input array.

Usage

result = STDEV(array [, mean])

Input Parameters

array — The array to be processed. Can be of any data type except string.

Output Parameters

mean — The mean of array.

Returned Value

result – The standard deviation of array.

Keywords

None.

Discussion

Mean and standard deviation are basic statistical tools used in a variety of applications. They are computed as follows:

$$mean = \frac{\sum_{i=0}^{n} array_i}{n}$$

where n is the number of elements in array.

$$STD = \sqrt{\frac{\sum_{i=0}^{n} (array_i - mean)^2}{n-1}}$$

In PV-WAVE, these equations are implemented as follows:

Example

$$y = [1, 5, 9, 3, 10, 4]$$

Create a simple array.

Compute the standard deviation and mean array value.

Print the results.

See Also

AVG, SIGMA

STOP Procedure

Stops the execution of a running program or batch file, and returns control to the interactive mode.

Usage

STOP [,
$$expr_1$$
, ..., $expr_n$]

Input Parameters

 $expr_i$ — One or more expressions whose value is printed. If no parameters are present, a brief message describing where the STOP was encountered is printed.

Keywords

None.

See Also

BREAKPOINT, RETALL, RETURN

STRARR Function

Returns a string array.

Usage

```
result = STRARR(dim_1, ..., dim_n)
```

Input Parameters

dim_i - The dimensions of the result. May be any scalar expression. Up to eight dimensions may be specified.

Returned Value

result — A string array.

Keywords

None.

Example

```
r = STRARR(2)
r(0) = 'one'
r(1) = 'two'
PRINT, r
   one two
```

See Also

BYTARR, COMPLEXARR, DBLARR, FLTARR, INTARR, LONARR, MAKE_ARRAY, SINDGEN, STRLEN

STRCOMPRESS Function

Compresses the white space in an input string.

Usage

result = STRCOMPRESS(string)

Input Parameters

string — The string to be compressed.

Returned Value

result — A string with all white space (blank spaces and tabs) compressed to a single space or completely removed. If string is an array, the result is an array with the same structure—each element contains a compressed copy of the corresponding element of string.

Input Keywords

Remove_All — If nonzero, removes all white space. Otherwise, all white space is compressed to a single space (the default).

Discussion

If not of type string, *string* is converted to string using the default formatting rules of PV-WAVE CL. (These rules are described in *Free Format Output* on page 164 of the *PV-WAVE Programmer's Guide*.)

Example 1

In this example, STRCOMPRESS is used to remove the excess white space in a string.

```
extra white space'
s = ' This string
                        has
   Create a string with excess white space characters.
PRINT, STRCOMPRESS(s)
    Display the string with all excess white space removed.
```

Example 2

This example uses STRCOMPRESS to remove all white space from each element of a three-element string array.

This string has extra white space

Create a three-element string array.

$$s(0) = 'a$$
 b c'
 $s(1) = '$ d e f'
 $s(2) = 'g$ h i'

s = strarr(3)

Assign a string to each element of the array.

PRINT, TRANSPOSE(STRCOMPRESS(s, /Remove All)) Remove all spaces from each element of the array and display the resulting array. TRANSPOSE is used to display the array as a column vector.

abc def qhi

See Also

STRTRIM, STRLEN, STRPOS, STRPUT

For an example, see Removing White Space From Strings on page 128 of the PV-WAVE Programmer's Guide.

STRETCH Procedure

Standard Library procedure that linearly expands the range of the color table currently loaded to cover an arbitrary range of pixel values.

Usage

STRETCH, low, high

Input Parameters

low — The lowest pixel value in the selected range; in other words, this is the pixel value that will be displayed with color index 0. The default value is 0.

high — The highest pixel value in the selected range; in other words, this is the pixel value that will be the highest color index available on a particular device (normally 255 on an eight-bit plane device). The default value is !D.N_Colors – 1.

Keywords

None.

Discussion

STRETCH linearly interpolates new Red, Green, and Blue color vectors between *low* and *high*, and then loads them into the image display with LOADCT. The color vectors in the PV-WAVE COLORS common block are unchanged.

For example, the command:

```
STRETCH, 100, 150
```

expands the color table so that the pixels in the range of 100 to 150 fill the entire color-intensity range.

To revert to a normal color table, call STRETCH with no parameters.

Example 1

```
OPENR, unit, !Data dir + 'mandril.img',$
   /Get_lun
READU, unit, mandril
FREE LUN, unit
   Read the PV=WAVE mandril demo image.
WINDOW, /Free, Colors=128, XSize=512, $
   YSize=512
TVSCL, mandril
   Display the original image using 128 colors.
LOADCT, 4
COLOR PALETTE
   Change the color palette and display it for reference.
!Err = 0
WHILE !Err LE 0 DO BEGIN
   CURSOR, x, y, /Change, /Normal
   range = FIX(y * 128.0 + 5.0)
   STRETCH, 64-range, 64+range
ENDWHILE
   Use the position of the cursor to compress or expand the palette
   about the middle using the STRETCH function.
```

mandril = BYTARR(512, 512)

Example 2

A simplified version of STRETCH can be written as:

```
PRO STRETCH, lo, hi
   Procedure definition.
COMMON colors, r orig, g orig, b orig, $
   r curr, g curr, b curr
       Access the color table common block used by PV=WAVE
       procedures.
t = BYTSCL(INDGEN(256), Min = lo, Max = hi)
```

Make a vector of subscripts into the color table arrays.

TVLCT, $r_orig(t)$, $g_orig(t)$, $b_orig(t)$ Load colors from the original tables transformed by t.

RETURN

END

See Also

!D.N_Colors, LOADCT, MODIFYCT, TVLCT, WgCtTool

For more information about how to access the three color variables in the color table common block, see *Retrieving Information About the Current Color Table* on page 321 of the *PV*-WAVE User's Guide.

For more information about stretching color tables, see *Stretching* the Color Table on page 319 of the PV-WAVE User's Guide.

STRING Function

Converts the input parameters to characters and returns a string expression.

Usage

$$result = STRING(expr_1, ..., expr_n)$$

Input Parameters

 $expr_i$ — The expressions to be converted to type string.

Returned Value

result — The string value for expr.

Input Keywords

Format – Allows the format of the output to be specified in precise detail, using a FORTRAN-style specification. FORTRANstyle formats are described in Appendix A, FORTRAN and C Format Strings, in the PV-WAVE Programmer's Guide.

Print — If present and nonzero, specifies that any special case processing should be ignored, and that STRING should behave exactly as the PRINT procedure.

Discussion

STRING can be used to convert one or more expressions to string data type. You can use the *Format* keyword to explicitly specify the format in the converted string. The format specification string used with the Format keyword follows this syntax:

$$"(q_1f_1s_1f_2s_2...f_nq_n)"$$

where:

- q is an optional slash (/) record terminator. During output, each record terminator causes the output to move to a new line. During input, each record terminator causes the next line of input to be read.
- f is a FORTRAN-style format string. Some format strings specify how data should be transferred while others control some other function related to how I/O is handled. If you do not specify a format string, the result is formatted automatically.

For more information about the FORTRAN format codes that can be used, refer to Appendix A, FORTRAN and C Format Strings, in the PV-WAVE Programmer's Guide.

s — is a comma (,) or a slash (/) that is used to separate input values. The only restriction on these separators is that two commas cannot occur side by side.



The quotation marks and parentheses surrounding the format string are required. The quotation marks can be either single or double quotation marks.

If the *Format* keyword is not present, PV-WAVE uses its default rules for formatting the output. These rules are described in Table 8-7 on page 164 of the *PV-WAVE Programmer's Guide*.

STRING is similar to the PRINT procedure, except that its output is placed in a string rather than being output to the terminal.

Example 1

The following three commands all yield the string scalar 'ABC':

```
x = STRING([65B,66B,67B])
y = STRING([byte('A'), byte('B'), byte('C')])
z = STRING('A' + 'B' + 'C')
```

In the first expression above, the input is a byte vector, and STRING converts it into a string scalar. The first parameter is 65B, which is a notation indicating that the parameter uses a byte data type, but the result is equal to the decimal ASCII code 65.

Example 2

STRING can also be used with a floating-point variable, as shown below.

```
x = 123.45
result = STRING(x)
PRINT, result
results in the output:
123.45
```

Example 3

Here are some examples using the *Format* keyword. In all three cases, the STRING command that is entered is:

```
result = STRING(numbers, Format="(I2)")
```

In these examples, the *Format* keyword instructs the STRING function to convert numbers to a string data type, using an integer format with a maximum field width of 2 characters.

If numbers = [36, 9, 72], and result is computed with the format string shown above, then entering the following command:

PRINT, result

results in the output:

36 9 72

The values are converted to strings, using a maximum field width of two characters.

If numbers = [5, 7, 9, 100], and result is computed with the format string shown above, then entering the following command:

PRINT, result

results in the output:

9 ** 5 7

The two asterisks indicate that the integer value 100 was too big to be read with an I2 FORTRAN format.

If numbers = [6.12, 4.507, 4.339], and result is computed with the format string shown above, then entering the following command:

PRINT, result

results in the output:

6 4 4

As shown in this example, an integer format can be used to output floating point data, but all digits after the decimal point are lost.

See Also

PRINT, STRARR, XYOUTS

For more information on string formats, see Chapter 7, Working with Text, in the PV-WAVE Programmer's Guide.

For more information on format specification codes, see Appendix A, FORTRAN and C Format Strings, in the PV-WAVE Programmer's Guide.

STRLEN Function

Returns the length of the input parameter.

Usage

```
result = STRLEN(expr)
```

Input Parameters

expr – The expression for which the string length is desired.

If not of type string, *expr* is converted to string using the default formatting rules of PV-WAVE CL. (These rules are described in *Free Format Output* on page 164 of the *PV-WAVE Programmer's Guide*.)

Returned Value

result — A long integer containing the string length of *expr*. If *expr* is an array, the result is an array with the same structure, with each element containing the length of the corresponding *expr* element.

Keywords

None.

Example

This example uses STRLEN to determine the length of several different strings.

```
a = 'a b c'
Create a scalar string.
PRINT, STRLEN(a)
Display the length of a.
```

```
b = 45
    Create an integer scalar variable.
PRINT, STRLEN(b)
    Display the length of b converted to string type.
8
c = STRING(FINDGEN(4), Format = '(f3.1)')
    Create a four-element vector of strings.
PRINT, TRANSPOSE(c)
    Display the contents of c as a column vector.
0.0
1.0
2.0
3.0
PRINT, TRANSPOSE(STRLEN(C))
    Display the vector of lengths of elements of c as a column vector.
    This vector is returned by STRLEN.
3
3
3
3
```

See Also

STRMID, STRPOS, STRPUT, STRTRIM

For an example, see Determining the Length of Strings on page 130 of the PV-WAVE Programmer's Guide.

STRLOWCASE Function

Converts a copy of the input string to lowercase letters.

Usage

result = STRLOWCASE(string)

Input Parameters

string — The string to be converted.

If not of type string, *string* is converted to string using the default formatting rules of PV-WAVE CL. (These rules are described in *Free Format Output* on page 164 of the *PV-WAVE Programmer's Guide*.)

Returned Value

result — A copy of *string* converted to lowercase letters. If *string* is an array, the result is an array with the same structure, with each element containing the substring of the corresponding *string* element.

Keywords

None.

Example

This example uses STRLOWCASE to convert uppercase characters to lowercase in several different strings.

a = 'A StRInG OF mIXeD CaSe' Create a string with a mix of uppercase and lowercase characters.

PRINT, STRLOWCASE(a)

a string of mixed case

Convert the string in a to lowercase and display the result.

```
b = 45
```

Create an integer scalar variable.

INFO, STRLOWCASE(b)

Examine the result of STRLOWCASE applied to b. Note that b is converted to a string.

```
= '
                                      45'
<Expression>
                  STRING
```

c = STRARR(3)

Create a three-element string vector.

```
c(0) = 'StrInG 0'
```

$$c(1) = 'sTrINg 1'$$

$$c(2) = 'StrinG 2'$$

Assign a string with a mix of uppercase and lowercase characters to each element of C.

PRINT, TRANSPOSE(STRLOWCASE(c))

Display the vector of strings of c after converting them to lowercase. Use Transpose to view the vector as a column.

```
string 0
```

string 1

string 2

Discussion

Only uppercase characters are modified by STRLOWCASE; lowercase and nonalphabetic characters are copied without change.

See Also

STRUPCASE, STRING, PRINT, MESSAGE, XYOUTS

For an example, see Converting Strings to Upper or Lower Case on page 127 of the PV-WAVE Programmer's Guide.

STRMESSAGE Function

Returns the text of the error message specified by the input error number.

Usage

result = STRMESSAGE(errno)

Input Parameters

errno — The error number for which the message text is desired.

Returned Value

result — The text of the error message specified by errno.

Keywords

None.

Discussion

This function is especially useful in conjunction with the !Err system variable, which always contains the error number of the last error.

However, you should take care not to make the assumption in your programs that certain error numbers always correspond to certain error messages, as this correspondence may change as PV-WAVE CL is modified over time.

Example

The following procedure uses function STRMESSAGE to display the error message associated with an input/output error trapped by procedure ON_IOERROR.

FUNCTION READ_DATA, file_name

Define a function to read, and return a 100-element, floating-point array.

ON IOERROR, bad

Declare error label.

OPENR, unit, file name, /Get Lun Open the file and use the Get_Lun keyword to allocate a logical file unit.

a = FLTARR(100)

Define data array.

READU, unit, a Read data.

GOTO, DONE

Clean up and return.

bad:

Exception label.

PRINT, STRMESSAGE (!ERROR)

Print the error message associated with the error number in !ERROR. Note that STRMESSAGE is used to display the message.

DONE:

FREE LUN, unit

Close and free the input/output unit.

RETURN, a

Return the result. Variable a is undefined if an error occurred.

END

See Also

!Err, !Err String, ON ERROR, ON_IOERROR, MESSAGE, **PRINT**

For more information on error handling, see Chapter 10, Programming with PV-WAVE, in the PV-WAVE Programmer's Guide.

STRMID Function

Extracts a substring from a string expression.

Usage

result = STRMID(expr, first character, length)

Input Parameters

expr — The expression from which the substring is to be extracted.

If not of type string, *expr* is converted to string using the default formatting rules of PV-WAVE CL. (These rules are described in *Free Format Output* on page 164 of the *PV-WAVE Programmer's Guide*.)

first_character — The starting position within *expr* at which the substring starts. The first character position is position 0.

length — The length of the substring. If there are not enough characters left in the main string to obtain *length* characters, the substring will be truncated.

Returned Value

result — A string of *length* characters taken from *expr*, starting at character position *first_character*. If *expr* is an array, the result is an array with the same structure, with each element containing the substring of the corresponding *expr* element.

Keywords

None.

Example

In this example, STRPOS is used to locate the first occurrence of the string a within a larger string. Function STRMID is then used to extract a substring from that position. Function STRPOS is used to locate the second occurrence of the string a, and finally, STR-MID is used to extract another substring from this second position within the larger string.

```
a = 'Extract a substring from a string'
   Create a string in the variable a.
POS = STRPOS(a, 'a')
    Locate first occurrence of the string "a".
PRINT, STRMID(a, pos, 11)
    Extract 11 characters from a, starting at POS.
a substring
POS = STRPOS(a, 'a', pos + 1)
    Search for the second occurrence of the string "a" by searching
   from the character position that is one greater than the first
   occurrence of that string. Extract 8 characters from a, starting at
   the new POS.
PRINT, STRMID(a, pos, 8)
a string
```

See Also

STRLEN, STRPOS, STRPUT

For an example, see Manipulating Substrings on page 131 of the PV-WAVE Programmer's Guide.

STRPOS Function

Searches for the occurrence of a substring within an object string, and returns its position.

Usage

result = STRPOS(object, search string [, position])

Input Parameters

object – The expression in which to search for the substring.

search_string — The substring to be searched for within object.

If object or search_string are not of type string, they are converted to string using the default formatting rules of PV-WAVE CL. (These rules are described in Free Format Output on page 164 of the PV-WAVE Programmer's Guide.)

position — The character position at which the search is begun. If **position** is omitted or is less than zero, the search begins at the first character (character position 0).

Returned Value

result — If search_string occurs in object, STRPOS returns the character position of the match; otherwise, it returns –1.

If *object* is an array, *result* is an array of the same structure, and each element contains the position of the substring within *object*.

If search_string is the null string, STRPOS returns the smaller of position or one less than the length of object.

Keywords

None.

Example

See the example for STRMID.

See Also

STRPUT, STRLEN, STRMESSAGE, STRLOWCASE, **STRUPCASE**

For an example, see Manipulating Substrings on page 131 of the PV-WAVE Programmer's Guide.

STRPUT Procedure

Inserts the contents of one string into another.

Usage

STRPUT, destination, source [, position]

Input Parameters

destination — The string into which source will be inserted. Must be a named variable of type string. Cannot be an element of an array. If destination is an array, source is inserted into every element.

source — The string to be inserted into destination. Must be scalar.

If not of type string, source is converted to string using PV-WAVE CL's default formatting rules. (These rules are described in Free Format Output on page 164 of the PV-WAVE Programmer's Guide.)

position — The character position at which the insertion is begun. If position is omitted or is less than zero, the search begins at the first character (character position 0).

Keywords

None.

Discussion

The source string is inserted into the destination string, starting at the given position. All characters in destination that occur either before the starting position, or after the starting position plus the length of source, remain unchanged.

Example

This example uses STRPUT to insert a substring into a larger string, starting at position 0 of the destination string. Clipping is then demonstrated by using STRPUT to insert a substring that would otherwise extend the length of the destination string.

```
a = 'Strings are fun'
    Create a string in the variable a.

STRPUT, a, 'PV-WAVE is' into a, starting at position 0.

PRINT, a
    Display the result.

PV-WAVE is fun

STRPUT, a, 'fun to use', 12
    Insert a string into a that would extend its length.

PRINT, a
    Display the result. Note that the inserted string was clipped so the length of a did not change.

PV-WAVE is fun
```

See Also

STRPOS, STRLEN, STRMESSAGE, STRLOWCASE, STRUPCASE

For an example, see *Manipulating Substrings* on page 131 of the *PV*-WAVE Programmer's Guide.

STR TO DT Function

Converts date and time string data to PV-WAVE Date/Time values.

Usage

result = STR TO DT(date strings [, time_strings])

Input Parameters

date_strings - A string constant or string array that contains date strings.

time_strings - A string constant or string array that contains time strings.

Returned Value

result - A PV-WAVE Date/Time variable containing the converted Date/Time data.

Input Keywords

Date_Fmt - Specifies the format of the date data in the input variable. Possible values are 1, 2, 3, 4, or 5, as summarized in the following table:

Table 2-11: Valid Date Formats

Value	Format Description	Examples for May 1, 1992
1	MM*DD*[YY]YY	05/01/92
2	DD*MM*[YY]YY	01-05-92
3	ddd*[YY]YY	122,1992
4	DD*mmm[mmmmmm]*[YY]YY	01/May/92
5	[YY]YY*MM*DD	1992-05-01

where the asterisk (*) represents one of the following separators: dash (-), slash (/), comma (,), period (.), or colon (:).

 $Time_Fmt$ — Specifies the format of the time portion of the data in the input variable. Possible values are -1 or -2, as summarized in the following table:

Table 2-12: Valid Time Formats

Value	Format Description	Examples for 1:30 p.m.
-1	HH*Mn*SS[.SSSS]	13:30:35.25
-2	HHMn	1330

where the asterisk (*) represents one of the following separators: dash (-), slash (/), comma (,), or colon (:). No separators are allowed between hours and minutes for the -2 format. Both hours and minutes must occupy two spaces.

For a detailed description of the date and time formats, see *Converting Your Data into Date/Time Data* on page 229 of the *PV*-WAVE User's Guide.



Date and time separators are specified with the !Date_Separator and !Time_Separator system variables. The STR_TO_DT function only recognizes the standard separators listed above. If any other separator is specified, this function does not work as expected.

Discussion

You can convert just date strings, just time strings, or both. If you do not pass in a date string, the resulting date portion of the Date/Time structure defaults to the value in the system variable !DT_Base. If you do not pass in a time string, the time portion of the resulting Date/Time variable defaults to zero.

Example 1

```
x = ['3-13-92', '4-20-83', '4-24-64']
   Create an array that contains some date strings with the MM DD
   YY date format.
```

```
y = ['1:10:34', '16:18:30', '5:07:25']
   Create an array that contains some time strings with the HH Mn
   SS date format.
```

```
date1 = STR_TO_DT(x, y, Date_Fmt=1, $
   Time Fmt =-1)
       Use the formats1 and -1 to return PV=WAVE Date/Time
       data.
```

```
DT PRINT, date1
   3/13/1992 01:10:34
   4/20/1983 16:18:30
   4/24/1964 05:07:25
```

Example 2

```
date2 = STR TO DT('3-13-92', Date_Fmt=1)
PRINT, date2
   { 1992 3 13 0 0 0.00000 87474.000 0 }
```

See Also

For more information on using Date/Time functions, see Chapter 7, Working with Date/Time Data, in the PV-WAVE User's Guide.

STRTRIM Function

Removes extra blank spaces from an input string.

Usage

result = STRTRIM(string [, flag])

Input Parameters

string — The string that will have leading and/or trailing blanks removed.

If not of type string, *string* is converted to string using the default formatting rules of PV-WAVE CL. (These rules are described in *Free Format Output* on page 164 of the *PV-WAVE Programmer's Guide*.)

flag — Controls the action of STRTRIM:

- If zero or not present, removes trailing blanks.
- If 1, removes leading blanks.
- If 2, removes both leading and trailing blanks.

Returned Value

result — A copy of *string* with extra (leading and/or trailing) blank spaces removed. If *string* is an array, the result is an array with the same structure — each element contains a trimmed copy of the corresponding element of *string*.

Keywords

None.

Example

In this example, STRTRIM is used to trim trailing, leading, then trailing and leading blanks from a string.

Create a string with both leading and trailing blanks.

Examine the contents of a. Note the presence of both leading and trailing blanks.

$$b = STRTRIM(a)$$

Remove trailing blanks from a.

INFO, b

Examine the results. Note that all trailing blanks are removed.

$$b = STRTRIM(a, 1)$$

Remove leading blanks from a.

INFO, b

Examine the results. Note that all leading blanks are removed.

$$b = STRTRIM(a, 2)$$

Remove both leading and trailing blanks from a.

INFO, b

Examine the results. Note that all leading and trailing blanks are removed.

See Also

STRCOMPRESS

For an example, see Removing White Space From Strings on page 128 of the PV-WAVE Programmer's Guide.

STRUCTREF Function

Returns a list of all existing references to a structure.

Usage

result = STRUCTREF({structure})

Input Parameters

structure — The structure name. The name can be specified as $\{structure\}$, "structure", or x, where x is a variable of type structure.

Returned Value

result — A list of the variables, structures, definitions, and common blocks that reference *structure*.

Keywords

None.

Discussion

Use STRUCTREF before you use DELSTRUCT to check if the structure you want to delete is currently referenced by any variables, common blocks, or other structure definitions.

You cannot delete a structure that is currently referenced.

If you want to delete a structure that is referenced, you can either:

- Delete the references (variables, structures, etc.) returned by STRUCTREF, and then use the DELSTRUCT procedure.
- Use the DELSTRUCT procedure with the *Rename* keyword. This renames the structure.

If you are trying to free memory, then you must pursue the first option. If you want to delete a structure, and memory is not a concern, then the second option is probably the best choice.

Examples

```
WAVE> x = \{struct1, a:float(0)\}
WAVE> PRINT, structref(x)
   cedure $MAIN$, symbol X>
```

See Also

DELSTRUCT, DELFUNC, DELPROC, N_TAGS, TAG_NAMES

For more information on structures, see Chapter 6, Working with Structures, in the PV-WAVE Programmer's Guide.

STRUPCASE Function

Converts a copy of the input string to uppercase letters.

Usage

result = STRUPCASE(string)

Input Parameters

string — The string to be converted.

If not of type string, *string* is converted to string using the default formatting rules of PV-WAVE CL. (These rules are described in *Free Format Output* on page 164 of the *PV-WAVE Programmer's Guide*.)

Returned Value

result — A copy of *string* converted to uppercase letters. If *string* is an array, the result is an array with the same structure, with each element containing the substring of the corresponding *string* element.

Keywords

None.

Discussion

Only lowercase characters are modified by STRUPCASE; uppercase and nonalphabetic characters are copied without change.

Example

This example uses STRUPCASE to convert lowercase characters to uppercase in several different strings.

```
a = 'A StRInG oF mixeD casE'
print, STRUPCASE(a)
```

A STRING OF MIXED CASE

Create a string with a mix of uppercase and lowercase characters. Convert the string in a to uppercase and display the result.

b = 45

INFO, STRUPCASE(b)

Create an integer scalar variable. Examine the result of STRUPCASE applied to b. Note that B is converted to a string.

```
45'
<Expression>
                   STRING
```

c = STRARR(3)

Create a three-element string vector.

```
c(0) = 'StrInG 0'
```

c(1) = 'sTrINg 1'

c(2) = 'StrinG 2'

Assign a string with a mix of uppercase and lowercase characters to each element of c.

```
PRINT, TRANSPOSE(STRUPCASE(C))
```

Display the vector of strings of c after converting it to uppercase. Use TRANSPOSE to view the vector as a column.

STRING 0

STRING 1

STRING 2

See Also

STRLOWCASE. STRING, PRINT, MESSAGE, XYOUTS

For examples, see Converting Strings to Upper or Lower Case on page 127 of the PV-WAVE Programmer's Guide.

SURFACE Procedure

Draws the surface of a two-dimensional array with hidden lines removed.

Usage

SURFACE, z [, x, y]

Input Parameters

z — A two-dimensional array containing the values that make up the surface. If x and y are supplied, the surface is plotted as a function of the X,Y locations specified by their contents. Otherwise, the surface is generated as a function of the array index of each element of z.

x - A vector or two-dimensional array specifying the X coordinates for the contour surface.

- If x is a vector, each element of x specifies the X coordinate for a column of z. For example, x(0) specifies the X coordinate for z(0, *).
- If x is a two-dimensional array, each element of x specifies the X coordinate of the corresponding point in z (x_{ij} specifies the X coordinate for z_{ii}).

y - A vector or two-dimensional array specifying the Y coordinates for the contour surface.

- If y is a vector, each element of y specifies the Y coordinate for a row of z. For example, y(0) specifies the Y coordinate for z (*, 0).
- If y is a two-dimensional array, each element of y specifies the Y coordinate of the corresponding point in z (y_{ij} specifies the Y coordinate for z_{ii}).

Keywords

SURFACE keywords are listed below. For a description of each keyword, see Chapter 3, Graphics and Plotting Keywords.

Ax	Normal	XTickformat	YTitle
Az	Position	XTicklen	
Background	Save	XTickname	ZAxis
Bottom	Skirt	XTicks	ZCharsize
Channel	Subtitle	XTickv	ZGridstyle
Charsize	T3d	XTitle	ZMargin
Charthick	Thick		ZMinor
Clip	Tickformat	YCharsize	ZRange
Color	Ticklen	YGridstyle	ZStyle
Data	Title	YMargin	ZTickformat
Device	Upper_Only	YMinor	ZTicklen
Font		YRange	ZTickname
Horizontal	XCharsize	YStyle	ZTicks
Linestyle	XGridstyle	YTickformat	ZTickv
Lower_Only	XMargin	YTicklen	ZTitle
Noclip	XMinor	YTickname	ZValue
Nodata	XRange	YTicks	
Noerase	XStyle	YTickv	

Discussion

Note the following restrictions on the use of SURFACE. If the X,Y grid is not regular or nearly regular, errors in hidden line removal will occur.

If the T3D keyword is set, the 3D to 2D transformation matrix contained in !P.T must project the Z axis to a line parallel to the device Y axis, or errors will occur.

Example

This example displays the surface described by the function

$$f(x,y) = x\sin(y) + y\cos(x)$$

where

$$(x, y) \in \{ \mathbb{R}^2 | x, y \in [-10, 10] \}$$

.RUN

- FUNCTION f, x, y
- RETURN, x * SIN(y) + y * COS(x)
- END

Define the function.

x = FINDGEN(101)/5-10
Create vector of x-coordinates.

y = x

Create vector of y-coordinates.

z = FLTARR(101, 101)

Create an array to hold the function values.

FOR
$$i = 0$$
, 100 DO FOR $j = 0$, 100 DO \$ $z(i, j) = f(x(i), y(j))$

Evaluate the function at the given x- and y-coordinates and place the result in z.

Display the surface. The Ax keyword is used to specify the angle of rotation about the x-axis. The XCharsize, YCharsize, and ZCharsize keywords are used to enlarge the characters used to annotate the axes.

Place a title on the window. Note that the CURSOR procedure with the Device keyword was used to locate the proper position for the title.

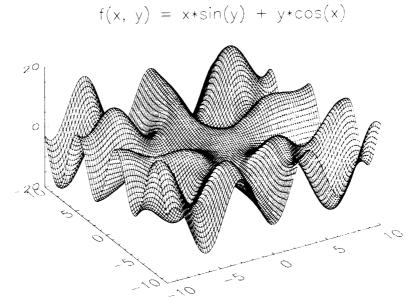


Figure 2-56 Surface with title.

See Also

!P.T, SHADE_SURF, SHADE_SURF_IRR, SURFACE_FIT, SURFR, T3D, THREED

For more information, see Chapter 4, Displaying 3D Data, in the PV-WAVE User's Guide.

SURFACE_FIT Function

Standard Library function that determines the polynomial fit to a surface.

Usage

result = SURFACE FIT(array, degree)

Input Parameters

array — The two-dimensional array of data to which the surface will be fit. The two dimensions do not have to be the same size. The number of data points must be $\geq (degree + 1)^2$.

degree - The maximum degree of the fit (in one dimension).

Returned Value

result — A two-dimensional array of values as calculated from the least-squares fit of the data.

Keywords

None.

Discussion

SURFACE_FIT first determines the coefficients of the polynomials in both directions, using the least-squares method of the POLYWARP function. It then uses these coefficients to calculate the data points of the *result* array.

Example

```
OPENR, 1, !Data dir + 'pikeselev.dat'
data = FLTARR(60, 40)
READF, 1, data
   Read in the data from the PV=WAVE pikes elevation file.
CLOSE, 1
   Close the input file.
SURFACE, data
   Show the raw data without fitting.
SURFACE, SURFACE_FIT(data, 1)
   Approximate a linear least-squares fit.
SURFACE, SURFACE FIT(data, 2)
   Do a second-degree polynomial fit.
SURFACE, SURFACE FIT(data, 3)
   Do a third-degree polynomial fit.
SURFACE, SURFACE FIT(data, 4)
   Do a forth-degree polynomial fit.
SURFACE, SURFACE_FIT(data, 5)
   Do a fifth-degree polynomial fit.
```

See Also

POLY_FIT, POLYWARP, SURFACE

SURFR Procedure

Standard Library procedure that duplicates the rotation, translation, and scaling of the SURFACE procedure.

Usage

SURFR

Parameters

None.

Input Keywords

Ax — The angle of rotation about the X axis in degrees. The default is 30 degrees.

Az — The angle of rotation about the Z axis in degrees. The default is 30 degrees.

Discussion

SURFR should be used for axonometric projections only. The 4-by-4 matrix in the system variable !P.T receives the homogeneous transformation matrix generated by this procedure.

SURFR performs the following steps:

the viewer.

30	Krk performs the following steps:
	Translates the unit cube so that the center (.5, .5, .5) is moved to the origin.
	Rotates -90 degrees about the X axis to make the +Z axis of the data the +Y axis of the display. The +Y data axis extends from the front of the display to the back.
	Rotates about the Y axis Az degrees. This rotates the result counterclockwise as seen from above the page.
	Rotates about the X axis Ax degrees, tilting the data towards

Translates back to the origin and scales the data so that the data are still contained within the unit cube after the transformation. (This step uses the SCALE3D procedure.)

Example 1

TEK COLOR

Load a color table.

data = HANNING(50, 50)

SURFR

Create the 3D spatial transformation. By default, the angle of rotation is defined as 30 degrees around the X and Y axes.

CONTOUR, data, Nlevels=20, /Follow, /T3D, \$ Charsize=1.5

Create the contour plot in the 3D space.

SURFR, Ax=45, Az=10

Now try it with different angles of rotation.

CONTOUR, data, Nlevels=20, /Follow, /T3D, \$ Charsize=1.5

Create the contour plot in the 3D space.



For an informative display, the above example could be done using the SHADE SURF or SURFACE procedures instead of SURFR, as shown below. (SURFR is not needed in this case because these two procedures themselves set up the 3D view area.)

SURFACE, data, /Save, Color=3 CONTOUR, data, Nlevels=20, /Follow, /T3D, \$ Charsize=1.5

Example 2

This example displays a cube as a shaded surface and rotates it.

```
xmin = 0 & xmax = 1
xmin = xmin - 1.5 \& xmax = xmax+1.5
!x.s = [-xmin, 1.0] / (xmax - xmin)
!y.s = !x.s
!z.s = !x.s
   Set up scaling for the 3D view.
vert = FLTARR(3, 8)
FOR i = 1, 2
                 DO vert(0, i) = 1
FOR i = 5, 6
                DO vert(0, i) = 1
FOR i = 2, 3 DO vert(1, i) = 1
FOR i = 6, 7
                DO vert(1, i) = 1
FOR i = 4, 7
                DO vert(2, i) = 1
   Create the array of vertices.
poly = FLTARR(30)
poly(0:4) = [4, 0, 3, 2, 1]
poly(5:9) = [4, 1, 2, 6, 5]
poly(10:14) = [4, 5, 6, 7, 4]
poly(15:19) = [4, 4, 7, 3, 0]
poly(20:24) = [4, 3, 7, 6, 2]
poly(25:29) = [4, 0, 1, 5, 4]
   Create the connectivity array.
SURFR
img = POLYSHADE(vert, poly, /T3D, /Data)
TV, imq
   Display the cube with the default rotations.
SURFR, Ax=50, Az=10
img = POLYSHADE(vert, poly, /T3D, /Data)
TV, img
   Display the cube rotated 50 degrees around the X axis and 10
   degrees around the Z axis.
```



This example can be done more simply with the CENTER VIEW procedure, which is available in the Advanced Rendering Library. This library is located in the wave/demo/arl directory. For more information, view the .pro file for this procedure.

See Also

!P.T, SCALE3D, SURFACE, T3D

SVBKSB Procedure

Uses "back substitution" to solve the set of simultaneous linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, given the u, w, and v arrays created by the SVD procedure from the matrix a.

Usage

SVBKSB, u, w, v, b, x

Input Parameters

- u The m-by-n column matrix of the decomposition of a, as returned by SVD.
- w The vector of singular values, as returned by SVD.
- v The *n*-by-*n* orthogonal matrix of the decomposition of a, as returned by SVD.
- b The vector containing the right-hand side of the equation.

Output Parameters

x - A variable containing the result.

Keywords

None.

Discussion

Since no input quantities are destroyed, SVBKSB may be called numerous times with different *b* vectors.

Example

Here's a typical use of SVD and SVBKSB to solve the system $\mathbf{A}\mathbf{x} = \mathbf{b}$:

```
SVD, A, w, u, v
Decompose square matrix A.

small = WHERE(w LT MAX(w) * 1.0e-6, count)
Get subscripts of singular values, using a given threshold.

IF count NE 0 THEN w(small) = 0.0
Zero singular values less than the threshold. See the Numerical Recipes book (referred to in the See Also section) for details.

SVBKSB, u, w, v, b, x
The vector x now contains the solution.

PRINT, TOTAL(ABS(A # x - b))
The residual, |Ax - b| should be near 0.
```

See Also

SVD

SVBKSB is based on the routine of the same name in *Numerical Recipes in C: The Art of Scientific Computing*, by Flannery, Press, Teukolsky, and Vetterling, Cambridge University Press, Cambridge, MA, 1988. It is used by permission.

SVD Procedure

Performs a singular value decomposition on a matrix.

Usage

SVD,
$$a$$
, $W[, u[, v]]$

Input Parameters

a — The two-dimensional matrix to be decomposed. It has m rows and n columns, and m must be greater than or equal to n.

Output Parameters

W - An n-element vector of "singular values" equal to the diagonal elements of w: $w_{i,j} = W_i$, $w_{i,j} = 0$ for $i \neq j$.

u — The m-by-n column matrix of the decomposition of a.

v — The *n*-by-*n* orthogonal matrix of the decomposition of a.

Keywords

None.

Discussion

SVD performs a singular value decomposition on a matrix a. It is useful for solving linear least-squares equations. SVD is able to deal with matrices that are singular or very close to singular.

The SVD procedure function transforms an m-by-n matrix a to the product of an m-by-n column orthogonal matrix u, an n-by-n diagonal matrix w, and the transpose of an n-by-n orthogonal matrix v. In other words, u, w, and v are matrices that are calculated by SVD.

By definition, the product of these three matrices (u, w, v) is equal to a:

$$a = uwv^{t}$$

In a linear system of equations,

$$Ax = b$$

where there are at least as many equations as unknowns, the least-squares solution vector x is given by:

$$x = v \cdot [diag(1/w_i)](u^t \cdot b)$$

Example

A PV-WAVE CL function that returns x, given A and b is:

```
Function SVD_SOLVE, A, b
   Return the vector x, the solution of the system of equations
   Ax = b. A is an (m, n) matrix, where m ≥ n.

SVD, A, w, u, v
   Call SVD to decompose A.

n = N_ELEMENTS(w)
   Make the diagonal matrix W<sub>i,i</sub> = 1/w<sub>i</sub>.

wp = FLTARR(n,n)

FOR i=0, n-1 DO IF w(i) NE 0 THEN $
   wp(i,i) = 1./w(i)

return, v # wp # (transpose(u) # b)
   Calculate x from equation and return.

END
```

See Also

SVBKSB

The SVD function is based on the subroutine SVDCMP in *Numerical Recipes in C: The Art of Scientific Computing*, by Flannery, Press, Teukolsky, and Vetterling, Cambridge University Press, Cambridge, MA, 1988, and is used by permission.

PV-WAVE's SVD is a translation of the ALGOL procedure SVD described in Linear Algebra, Vol. II of The Handbook for Automatic Computation, by Wilkinson and Reinsch, Springer-Verlag, New York, NY, 1971.

For more information on SVD, also consult Computer Models for Mathematical Computations, by Forsythe, Malcom, and Moler, Prentice Hall, Englewood Cliffs, NJ, 1977.

SVDFIT Function

Standard Library function that uses the singular value decomposition method of least-squares curve fitting to fit a polynomial function to data.

Usage

result = SVDFIT(x, y, m)

Input Parameters

x - A vector containing the X (independent) coordinates of the data.

y - A vector containing the Y (dependent) coordinates of the data. Should have the same number of elements as x.

m — The number of coefficients in the fitted function. For polynomials, m is one more than the degree of the polynomial.

Returned Value

result — A vector containing the coefficients of the polynomial equation which best approximates the data. It has a length of m.

Input Keywords

Weight — A vector of weighting factors for determining the weighting of the least-squares fit. Must have the same number of elements as x.

Output Keywords

Chisq — The sum of the squared errors, multiplied by the weights, if *Weight* is specified.

Y fit — A vector containing the calculated Y values of the fitted function.

Singular — The number of singular values (i.e., the number of values that are inconsistent with the other data) encountered in evaluating the fit. Should be 0; if not, the computed polynomials probably do not accurately reflect the data.

Variance — The estimated variance (sigma squared) for each of the *m* coefficients.

Covar — The covariance matrix of the *m* coefficients.

Funct — The name of a user-supplied basis function with m coefficients (see *User-Supplied Basis Function* section below).

Discussion

You must define any keywords to be returned by SVDFIT before calling it. The value or structure of the variable doesn't matter, since the variable will be redefined dynamically in the function call. For example:

```
yf = 1
    Define the output variable yf.
c = SVDFIT(x, y, m, Yfit=yf)
    Do the SVD fit and return the Yfit vector in the variable yf.
```

User-Supplied Basis Function

If *Funct* is not supplied, a polynomial function is used as the basis function. This function is of the form:

$$result(i, j) = x(i) ^ j$$

The function is called with two parameters in the following fashion:

$$result = FUNCT(x, m)$$

where x and m are as defined above.

The file containing Funct should reside in the current working directory or in the search path defined by the system variable !Path.

If Funct does not have the file extension .pro, it should be compiled before it is called in SVDFIT.

See the Standard Library function cosines.pro, which defines a basis function of the form:

$$result(i, j) = COS(j * x(i))$$

Weighting is useful when you want to correct for potential errors in the data you are fitting to a curve. The weighting factor, Weight, adjusts the parameters of the curve so that the error at each point of the curve is minimized. For more information, see the section Weighting Factor on page 149, in Volume 1 of this reference.

See Also

COSINES, FUNCT, REGRESS

For more examples of basis functions, see Numerical Recipes in C: The Art of Scientific Computing, by Flannery, Press, Teukolsky, and Vetterling, Cambridge University Press, Cambridge, MA, 1988.

SYSTIME Function

Returns the current system time as either a string or as the number of seconds elapsed since January 1, 1970.

Usage

result = SYSTIME(param)

Input Parameters

param — If present and nonzero, causes the number of seconds elapsed since January 1, 1970 to be returned as a double-precision floating-point value. Otherwise, a scalar string containing the current date/time in standard system format is returned.

Returned Value

result — Either a value representing the number of seconds since January 1, 1970 or a string containing the time in the standard system time format.

Keywords

None.

Example

```
t1 = SYSTIME(1)
   Calculate number of seconds elapsed since January 1, 1970.
t2 = SYSTIME(0)
   Calculate the current date/time in standard system format.
PRINT, t1, t2
   6.9248465e+08 Wed Dec 11 13:48:33 1991
```

See Also

For information on other ways that PV-WAVE can handle dates and times, see Chapter 7, Working with Date/Time Data, in the PV-WAVE User's Guide.

T3D Procedure

Standard Library procedure that accumulates one or more sequences of translation, scaling, rotation, perspective, or oblique transformations and stores the result in the system variable !P.T.

Usage

T₃D

Parameters

None.

Input Keywords

Reset — A scalar which, if nonzero, resets !P.T (the transformation matrix) back to the default identity matrix.

Translate - A three-element vector containing the specified translations in the X, Y, and Z directions.

Scale - A three-element vector containing the specified scaling factors in the X, Y, and Z directions.

Rotate - A three-element vector, in units of degrees, containing the specified rotations about the X, Y, and Z axes. Rotations are performed in the order of X, Y, and then Z.

Perspective — A scalar (p) indicating the Z distance to the center of the projection. Objects are projected onto the XY plane at Z=0, and the "eye" is located at point (0, 0, p).

Oblique - A two-element vector containing the oblique projection parameters. Points are projected onto the XY plane at Z=0 as follows:

$$X' = X + Z(d * cos(a))$$

$$Y' = Y + Z(d * sin(a))$$

where Oblique(0) = d and Oblique(1) = a

XYech — If nonzero, exchanges the X and Y axes.

XZech — If nonzero, exchanges the X and Z axes.

YZech — If nonzero, exchanges the Y and Z axes.

Discussion

All parameters are entered in the form of keywords. The transformation specified by each keyword is performed in the order of its description above. For example, if both *Translate* and *Scale* are specified, the translation is done first, even if *Scale* is the first keyword in the procedure call.

The 4-by-4 transformation matrix, !P.T, is updated by this procedure, but the system variable !P.T3D is not set. This means that for the transformations to take effect, you must set !P.T3D equal to 1, or use the *T3D* keyword, in any plotting procedure that will make use of this transformation. Since all the PV-WAVE CL graphic routines use the !P.T matrix for output, T3D can be used to effect the graphic output of PV-WAVE CL.

Caution |



It is possible to create a transformation matrix with T3D that will not work correctly with the SURFACE or SHADE_SURF procedures. The only T3D transformations allowed with these procedures are those that end up with the Z axis placed vertically on the display screen.

This procedure follows the example given in *Fundamentals of Interactive Computer Graphics*, by Foley and Van Dam, Addison Wesley Publishing Company, Reading, MA, 1982.

Note

The matrix notation used in the procedure is reversed from the normal PV-WAVE CL sense in order to conform to this reference. Moreover, a right-handed system is used, meaning that positive rotations are counterclockwise when looking from a positive axis to the origin.

Example 1

T3D, /Reset, Rotate=[30,0,0], Perspective=-1 Reset transformation, rotate 30 degrees about the X axis, and then do a perspective transformation with the center of the projection at (0, 0, -1).

Example 2

Transformations may also be cascaded. For example:

Reset, translate the center (.5, .5, 0) to the origin, and rotate 45 degrees counterclockwise about the Z axis.

Move the origin back to the center of the viewport.

Example 3

```
b = FINDGEN(37)*10
```

$$y = SIN(b*!Dtor)/EXP(b/200)*4$$

$$sz = SIZE(y)$$

$$j = REFORM(y, sz(1), 1)$$

Create the data, and reform it into a 2D array so that it can be used with the SURFACE command.

Draw a 3D axis and set up a 3D screen.

Exchange the Y and Z axes so that plots are stacked one in front of the other along the usual Y axis.

PLOT, b, y, /T3D, ZValue=1.0, YMargin=[0, 0],\$ /Noerase

Create a line plot stacked on the 3D axis.

POLYFILL, b, y, /T3D, Color=6, Z=1.0 Fill under the curve.

PLOT, b, y, /T3D, ZValue=0.5, YMargin=[0, 0],\$
 /Noerase

POLYFILL, b, y, /T3D, Color=8, Z=0.5
 Create and fill a second curve.

PLOT, b, y, /T3D, ZValue=0.0, YMargin=[0, 0],\$
 /Noerase

POLYFILL, b, y, /T3D, Color=7, Z=0.0

T3D, /Reset

Create and fill a third curve.

See Also

!P.T, !P.T3D, SURFR

For more information and examples, see *Drawing Three-dimensional Graphics* on page 115 of the *PV-WAVE User's Guide*.

TAG_NAMES Function

Returns a string array containing the names of the tags in a structure expression.

Usage

result = TAG NAMES(expr)

Input Parameters

expr — The expression for which the tag names will be returned. Must be of structure type.

Returned Value

result — The string array containing the names of the tags. If *expr* is a structure containing nested structures, only the names of tags in the outermost structure are returned (as TAG_NAMES does not search for tags recursively).

Keywords

None.

· Example

This example uses TAG_NAMES to display the tag names of a structure and one of its fields, which is also a structure.

```
a = {struc1, t1: 0.0D, t2: {struct2, $
   t2 t1: INTARR(3), t2_t2: 0.0, $
   t2 t3: OL }, t3: FLTARR(12), t4: OL }
        Create a structure containing four fields, the second of
        which is also a structure.
PRINT, TAG NAMES(a)
   Display tag names of a.
T1 T2 T3 T4
PRINT, TAG_NAMES(a.t2)
   Display tag names of the structure in the second field of a.
T2_T1 T2_T2 T2_T3
```

See Also

DELSTRUCT, STRUCTREF, N_TAGS

TAN Function

Returns the tangent of the input variable.

Usage

$$result = TAN(x)$$

Input Parameters

x — The angle, in radians, that is evaluated. Cannot be a complex data type.

Returned Value

result — The tangent of x.

Keywords

None.

Discussion

If x is of double-precision floating-point, TAN yields a result of the same data type. All other data types, except complex, yield a single-precision floating-point result.

If x is an array, the result of TAN has the same dimensions, with each element containing the tangent of the corresponding element of x.

Example

```
x = [-60, -30, 0, 30, 60]
PRINT, TAN(x * !Dtor)
-1.73205 -0.577350 0.00000 0.577350 1.73205
```

See Also

TANH, ATAN, COS, SIN

For a list of other transcendental functions, see Transcendental Mathematical Functions on page 20, in Volume 1 of this reference.

TANH Function

Returns the hyperbolic tangent of the input variable.

Usage

result = TANH(x)

Input Parameters

x — The angle, in radians, that is evaluated.

Returned Value

result — The hyperbolic tangent of x.

Keywords

None.

Discussion

TANH is defined by:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

If x is of double-precision floating-point or of complex data type, TANH yields a result of the same type. All other data types yield a single-precision floating-point result.

If x is an array, the result of TANH has the same dimensions, with each element containing the hyperbolic tangent of the corresponding element of x.

Example

See Also

COSH, SINH, TAN, ATAN

For a list of other transcendental functions, see *Transcendental Mathematical Functions* on page 20, in Volume 1 of this reference.

TAPRD Procedure

(VMS Only) Reads the next record on the selected tape unit into the specified array.

Usage

TAPRD, array, unit [, byte reverse]

Input Parameters

array — A named variable into which the data should be read. The length of array and the records on the tape may range from 14 to 65,235 bytes.

- If array is larger than the tape record, the last portion of array is not changed.
- If array is shorter than the tape record, a data overrun error occurs.

unit — A number between 0 and 9 specifying the magnetic tape unit to rewind. (Do not confuse this parameter with file logical unit numbers.)

byte_reverse - If present, causes the even and odd numbered bytes to be swapped after reading, regardless of the type of data or variables. This facilitates reading tapes written on IBM machines.

Keywords

None.

Discussion

No data or format conversion, with the exception of optional byte reversal, is performed by TAPRD. The input array must be defined with the desired type and dimensions.

If the read is successful, the system variable !Err is set to the number of bytes read.

See Also

SKIPF, TAPWRT

For more information, see Accessing Magnetic Tape on page 229 of the PV-WAVE Programmer's Guide.

TAPWRT Procedure

(VMS Only) Writes data from the input array to the selected tape unit.

Usage

TAPWRT, array, unit [, byte_reverse]

Input Parameters

array — The variable from which the data should be output. May be an expression. The length of array and the records on the tape may range from 14 to 65,235 bytes.

unit — A number between 0 and 9 specifying the magnetic tape unit to rewind. (Do not confuse this parameter with file logical unit numbers.)

byte_reverse — If present, causes the even and odd numbered bytes to be swapped on output, regardless of the type of data or variables. This facilitates writing tapes compatible with IBM machines.

Keywords

None.

Discussion

One physical record containing the same number of bytes as *array* is written each time TAPWRT is called.

See Also

SKIPF, TAPRD

For more information, see Accessing Magnetic Tape on page 229 of the PV-WAVE Programmer's Guide.

TEK_COLOR Procedure

Standard Library procedure that loads a color table, which contains 32 distinct colors and is similar to the default Tektronix 4115 color table, into the display.

Usage

TEK COLOR

Parameters

None.

Keywords

None.

Discussion

TEK COLOR loads the first 32 elements of the color table with the Tektronix 4115 default color map. This creates a useful color table if you desire distinctive colors.

Example 1

```
b = FINDGEN(37)
x = b * 10
y = SIN(x * !Dtor)
```

Create an array containing the values for a sine function from 0 to 360 degrees.

PLOT, x, y, XRange=[0,360], XStyle=1, YStyle=1 Plot data and set the range to be exactly 0 to 360.

```
COLOR PALETTE
```

Put up a window containing a display of the current color table and its associated color indices.

TEK COLOR

Load a predefined color table that contains 32 distinct colors.

```
POLYFILL, x, y, Color=6
POLYFILL, x, y/2, Color=3
POLYFILL, x, y/6, Color=4
```

Fill in areas under the curve with different colors.

```
z = COS(x * !Dtor)
```

Create an array containing the values for a COS function from 0 to 360 degrees.

```
OPLOT, x, z/8, Linestyle=2, Color=5
Plot the cosine data on top of the sine data.
```

Example 2

This example creates a contour plot of the PV-WAVE Pike's Peak elevation data file, with the area in between the contour lines filled with a solid color.

POLYCONTOUR, 'path.dat'

Display the contour plot with contours filled with solid colors.

pikes = REBIN(pikes, 600, 400)

Resize the original data so that it will be a good size when it is displayed.

TV, BYTSCL(pikes, Top=10)

Display the plot as an image similar to a polycontour. (Displaying the plot as an image gives you access to the various image processing and analysis routines, such as DEFROI, HISTOGRAM, and PROFILES.)

See Also

ADJCT, C_EDIT, COLOR_EDIT, COLOR_PALETTE, LOADCT, PALETTE, TVLCT

THREED Procedure

Standard Library procedure that plots a two-dimensional array as a pseudo three-dimensional plot on the currently selected graphics device.

Usage

THREED, array [, space]

Input Parameters

array — The two-dimensional array to plot.

space — The spacing between lines of the plot:

- If space is omitted, spacing will be set to (MAX(A) - MIN(A))/ROWS.
- If space is negative, no hidden lines will be removed.

Input Keywords

Title — A string containing the main plot title.

XTitle — A string containing the title of the X axis.

YTitle - A string containing the title of the Y axis.

Discussion

The orientation of the data plotted by THREED is fixed.

Example

```
OPENR, 1, !Data_dir + 'pikeselev.dat'
data = FLTARR(60, 40)
READF, 1, data
CLOSE, 1
THREED, data
SURFACE, data
```

See Also

SURFACE, T3D

TODAY Function

Returns a Date/Time variable containing the current system date

Usage

```
result = TODAY()
```

Parameters

None.

Returned Value

result - A PV-WAVE Date/Time variable containing the current system date and time.

Keywords

None.

Example

```
dttoday = TODAY()
PRINT, dttoday
   {1992 4 29 14 47 55.0000 87521.617 0}
DT PRINT, dttoday
   04/29/1992
               14:47:55
```

See Also

DTGEN

For more information, see Chapter 7, Working with Date/Time Data, in the PV-WAVE User's Guide.

TOTAL Function

Sums the elements of an input array.

Usage

```
result = TOTAL(array)
```

Input Parameters

array – The array that is totalled. Can be of any data type except string.

Returned Value

result — A scalar value equal to the sum of all the elements of array.

Keywords

None.

Discussion

If array is of double-precision floating-point or complex data type, the result is of the same type. If array is any other data type, TOTAL returns single-precision floating-point.

Example

In this example, TOTAL is used to compute the sums of all elements in various rows and columns of a 3-by-2 integer array.

$$a = INDGEN(3, 2)$$

Create a 3-by-2 integer array. Each element has a value equal to its one-dimensional subscript.

```
PRINT, TOTAL(a(*, 0))
   Display the sum of the elements in the first row.
3.00000
PRINT, TOTAL(a(1, *))
   Display the sum of the elements in the second column.
5.00000
PRINT, TOTAL(a)
   Display the sum of all elements in the array.
```

See Also

SIZE, AVG, MIN, MAX, MEDIAN

TQLI Procedure

Uses the QL algorithm with implicit shifts to determine the eigenvalues and eigenvectors of a real, symmetric, tridiagonal matrix.

Usage

TQLI, d, e, z

15.0000

Input Parameters

- d An n-element vector. On input, it contains the diagonal elements of the matrix.
- e An n-element vector. On input, it contains the off-diagonal elements of the matrix. e_0 is arbitrary.

Output Parameters

- d An n-element vector. On output, it contains the eigenvectors.
- e An n-element vector. On output, it is destroyed.

z — A matrix containing the n eigenvectors. The eigenvectors are stored by rows; for example, z (*, θ) contains the first eigenvector.

Keywords

None.

Discussion

The TRED2 procedure may be used to reduce a real symmetric matrix to the tridiagonal form that is suitable for input to TQLI.

If the eigenvectors of a tridiagonal matrix are desired, z should be input as an identity matrix. If the eigenvectors of a matrix that has been reduced by TRED2 are desired, z should be input as the matrix Q output by TRED2.

TQLI is based on the routine of the same name found in *Numerical Recipes in C: The Art of Scientific Computing*, by Flannery, Press, Teukolsky, and Vetterling, Cambridge University Press, Cambridge, MA, 1988. It is used by permission.



However, because the order of subscripts in PV-WAVE are reversed in comparison to those in *Numerical Recipes*, the *z* matrix is transposed from the result described in that book.

Example

To determine the eigenvalues and eigenvectors of a real, symmetric matrix A:

asave = A

Save original matrix, as TRED2 destroys its input.

TRED2, A, d, e

Reduce matrix A to tridiagonal form.

TQLI, d, e, A

Obtain n eigenvalues in d, and n eigenvectors in A(*, i).

To verify the operation of these routines, use the definition of eigenvalues and eigenvectors. Matrix A is said to have an eigenvector \mathbf{x} and corresponding eigenvalue λ if:

$$A \cdot x = \lambda x$$

This is demonstrated by the following code fragment:

FOR i=0, N_ELEMENTS(d)-1 DO BEGIN For each eigenvector/value:

PRINT, 'Eigenvalue', i, '=', d(i) Print the eigenvalue.

PRINT, 'Eigenvector = ', Aa(*,i) Print the computed eigenvector.

PRINT, 'A x / lambda = ', asave # A(*,i)/d(i)Print the eigenvector again. This row vector should be equal to the eigenvector printed above.

ENDFOR

See Also

TRED2

TRANSPOSE Function

Transposes the input array.

Usage

result = TRANSPOSE(array)

Input Parameters

array — The array to be transposed. Must have one or two dimensions.

Returned Value

result — The transposed array.

Keywords

None.

Discussion

TRANSPOSE provides a convenient way to convert a row vector into a column vector, or the reverse. For example, it can be used to change a two-dimensional array into a one-dimensional vector, or to change an *m-by-n* array into an *n-by-m* array.

Example 1

For example, executing the statements:

$$a = INDGEN(10)$$

a is a 10-element row vector.

$$b = TRANSPOSE(a)$$

b is a 10-element column vector.

Show the results.

gives the result:

a INT =
$$Array(10)$$

b INT =
$$Array(1, 10)$$

Example 2

TRANSPOSE can also be used to shift the elements of a square array around the diagonal. For example, suppose you have the following array:

Applying TRANSPOSE to this array will yield the following result:

^{2 1 3}

Example 3

Here is what an aerial image looks like before and after applying TRANSPOSE.





Figure 2-57 TRANSPOSE has been used with this 512-by-512 aerial image to flip it diagonally (rotate it and create a mirror image).

See Also

INVERT, ROT, ROTATE

TRED2 Procedure

Reduces a real, symmetric matrix to tridiagonal form, using Householder's method.

Usage

TRED2, a [, d [, e]]

Input Parameters

a - An n-by-n real, symmetric matrix.

Output Parameters

a — This input parameter is replaced, on output, by the orthogonal matrix Q, effecting the transformation. The TQLI procedure uses this result to find the eigenvectors of the matrix A.

d – An n-element vector containing the diagonal elements of the tridiagonal matrix.

e — An n-element vector containing the off-diagonal elements of the tridiagonal matrix.

Keywords

None.

See Also

TOLI

TRED2 is based on a routine of the same name in Numerical Recipes in C: The Art of Scientific Computing, by Flannery, Press, Teukolsky, and Vetterling, Cambridge University Press, Cambridge, MA, 1988. It is used by permission.

TRIDAG Procedure

Solves tridiagonal systems of linear equations.

Usage

TRIDAG, a, b, c, r, u

Input Parameters

a — An n-element vector containing n — 1 subdiagonal elements. a_0 is ignored.

b - An n-element vector containing n diagonal elements.

c — An n-element vector containing n-1 superdiagonal elements. c_{n-1} is ignored.

r — An n-element vector containing the right-hand side of the equation $A^{T}u = r$.

Output Parameters

 $u - \operatorname{An} n$ -element floating-point vector containing the solution for tridiagonal systems of linear equations.

Keywords

None.

Discussion

The input vectors a, b, c, and r are not modified by TRIDAG. They contain, respectively, the subdiagonal, diagonal, and superdiagonal elements of A, and the right-hand side of the equation:

$$A^{T}u = r$$

The solution is stored in the variable u.



Because PV-WAVE subscripts are in column-row order, the above equation is written as $A^{T}u = r$, rather than as Au = r.

See Also

LUBKSB, LUDCMP, MPROVE, SVBKSB

TRIDAG is based on a routine of the same name in Numerical Recipes in C: The Art of Scientific Computing, by Flannery, Press, Teukolsky, and Vetterling, Cambridge University Press, Cambridge, MA, 1988, and is used by permission.

TRNLOG Function

(VMS Only) Searches the VMS name tables for a specified logical name and returns the equivalence string(s) in a PV-WAVE variable.

Usage

result = TRNLOG(logname, value)

Input Parameters

logname — A scalar string containing the name of the logical to be translated.

Output Parameters

value - A variable into which the equivalence string is placed. If *logname* has more than one equivalence string, the first one is used.

Note that you can use the Full Translation keyword to obtain all equivalence strings.

Returned Value

result — The VMS status code associated with the translation as a longword value:

- An odd value (the least significant bit is set) indicates success.
- An even value indicates failure.

Input Keywords

Acmode — An integer specifying the access mode to be used in the translation. Valid values are listed below:

- 0 Kernel
- 1 Executive
- 2 Supervisor
- 3 User

If you use the *Acmode* keyword, all names at access modes less privileged than the specified mode are ignored.

If you don't use *Acmode*, the translation proceeds without regard to access mode. However, the search proceeds from the outermost (User) to the innermost (Kernel) mode. Thus, if two logical names with the same name but different access modes exist in the same table, the name with the outermost access mode is used.

Full_Translation — If present and nonzero, specifies that value should be turned into a string array. Each element of this array is one of the equivalence strings.

If Full_Translation is not present, value only receives the first equivalence string as a scalar value, when translating a multivalued logical name.

For example, under recent versions of VMS, the SYS\$SYSROOT logical can have multiple values. To see these values from within PV-WAVE:

```
ret = TRNLOG('SYS$SYSROOT', trans, /Full, $
    /Issue_Error)
    Translate the logical.
```

PRINT, trans

View the equivalence strings.

Issue_Error — If present and nonzero, causes TRNLOG to issue an error message if the translation fails.

Table - A scalar string giving the name of the logical table in which the search for logname will occur. If Table is not specified, the standard VMS logical tables are searched until a match is found, starting with LNM\$PROCESS TABLE and ending with LNM\$SYSTEM TABLE.

Output Keywords

Result Acmode — If present, specifies a named variable into which the access mode value of the translated logical will be placed. (The access mode values are summarized in the Input Keywords section.)

Result_Table — If present, specifies a named variable into which the name of the logical table containing the translated logical will be placed, as a scalar string.

See Also

DELETE SYMBOL, DELLOG, GET SYMBOL, SETLOG, SET SYMBOL

TV Procedure

Displays images without scaling the intensity.

Usage

TV, image [, x, y [, channel]]
TV, image [, position]

Input Parameters

image — A vector or two-dimensional matrix to be displayed as an image.

If *image* is not already of byte type, it is converted prior to use, although the conversion may distort the data contained in the image.



To insure data integrity, use the TVSCL procedure instead, which does a byte scaling before displaying the image.

x, y — The X and Y lower-left coordinates of the displayed image.

channel — The memory channel to be written. If not specified, it is assumed to be zero. This parameter is ignored on display systems that have only one memory channel.

position — Number specifying the position of the image. Positions run from the left of the window to the right, and from the top of the window to the bottom (see Discussion below for details).

Input Keywords

Centimeters — If present, specifies that all position values (the x y parameters and the XSize and YSize keywords) are in centimeters from the origin. This is useful when dealing with devices that do not provide a direct relationship between image pixels and the size of the resulting image, such as PostScript printers.

If *Centimeters* is not present, position values are taken to be in device coordinates

Channel — The memory channel to be written. This keyword is identical to the channel input parameter; only one needs to be used.

Data — If present, specifies that all position and size values are in data coordinates. This is useful when drawing an image over an existing plot, since the plot establishes the data scaling.

Device — If present, specifies that all position and size values are in device coordinates. This is the default.

Inches — If present, specifies that all position and size values are in inches from the origin. This is useful when dealing with devices that do not provide a direct relationship between image pixels and the size of the resulting image, such as PostScript printers.

Normal — If present, specifies that all position and size values are in normalized coordinates in the range 0.0 to 1.0. This is useful when you want to draw an image in a device-independent manner.

Order – If specified, overrides the current setting of the !Order system variable for the current image only. If nonzero, Order causes the image to be drawn from the top-down, instead of from the bottom-up (the default).

True — If present and nonzero, indicates that a true-color (24-bit) image is to be displayed and specifies the index of the dimension over which color is interleaved:

- 1 Displays pixel-interleaved images of dimensions (3, m, n).
- 2 Displays row-interleaved images of dimensions (m, 3, n).
- 3 Displays image-interleaved images of dimensions (m, n, 3). (Image interleaving is also known as band interleaving.)



To use True, the image parameter must have three dimensions, one of which is equal to 3.

XSize — The width of the resulting image. This keyword is intended for devices with scalable pixel size (such as PostScript), and is ignored by pixel-based devices that are unable to change the size of their pixels (such as a Sun Workstation monitor).

YSize — The height of the resulting image, with the same limitations as XSize.

Z — The Z position. The value of Z is of use only if the three-dimensional transformation is in effect via T3D.

Discussion

If position is used instead of the x and y parameters, the position of the image is calculated based on the largest grid of images of this size that will fit on the display, numbered from left to right and top to bottom. Specifically, the position is calculated as explained below.

The starting X coordinate position is defined as:

$$x = x_{dim} \cdot position_{moduloN_r}$$

The starting Y coordinate position is defined as:

$$y = YSize - y_{dim} \cdot int(1 + (position/N_x))$$

where

YSize is the height of the display or window,

 x_{dim} and y_{dim} are the dimensions of the array, and the images across the display surface are defined as:

$$N_x = \frac{Xsize}{Ysize}$$

For example, when displaying 128-by-128 images on a 512-by-512 display, the position numbers run from 0 to 15 as follows:

Example 1

```
mandril = BYTARR(512, 512)
OPENR, unit, !Data dir + 'mandril.img', $
   /Get_lun
READU, unit, mandril
FREE LUN, unit
   Read the PV=WAVE mandril demo image.
WINDOW, XSize=512, YSize=512
TV, mandril
   Display the mandril image.
TV, mandril, /Order
   Display the mandril image from bottom-up instead of top-down;
   this inverts the image.
small_img = CONGRID(mandril, 100, 100)
   Interpolate a smaller mandril image of size 100-by-100 pixels.
CONTOUR, small_img
TV, small img, 0, 0, /Data
   Create a contour plot of the data and overlay the image at the
   origin, specifying the image position in data coordinates.
FOR i=0, 24 DO TV, small img, i
   Tile the entire window with mandrils, using the position parame-
   ter rather than specifying X and Y locations.
```

Example 2

This example uses TV in conjunction with the REBIN and CONGRID procedures to enlarge small images.

```
data = BYTSCL(DIST(60))
   Create the data.
TV, data
   Display the data.
LOADCT, 5
   Load a color table.
```

enlarge = REBIN(data, 480, 480)
Enlarge the original data.

TV, enlarge

Display the area enlarged with REBIN.

enlarge = REBIN(data, 480, 480, /Sample)

TV, enlarge

Redisplay the area enlarged with REBIN using the nearest neighbor method for the sampling.

enlarge = CONGRID(data, 480, 480)

TV, enlarge

Display the area enlarged with CONGRID.

enlarge = CONGRID(data, 480, 480, /Interp)

TV, enlarge

Redisplay the area enlarged with CONGRID using the bilinear interpolation method for the sampling.

See Also

!Order, BYTSCL, TVCRS, TVLCT, TVRD, TVSCL

For more information, see *Image Display Routines: TV and TVSCL* on page 136 of the *PV*-WAVE User's Guide.

TVCRS Procedure

Manipulates the cursor within a displayed image, allowing it to be enabled and disabled, as well as positioned.

Usage

```
TVCRS [, on off]
TVCRS [, x, y]
```

Input Parameters

on off — Specifies whether the cursor should be on or off. If present and nonzero, enables the cursor. If zero or not specified, disables the cursor.

x — The column to which the cursor will be positioned.

y — The row to which the cursor will be positioned.

Input Keywords

Centimeters – If present, specifies that all position values are in centimeters from the origin.

Data – If present and nonzero, specifies that the cursor position is in data coordinates.

Device — If present and nonzero, specifies that the cursor position is in device coordinates.

Hide — If present and nonzero, causes a disabled cursor to always be blanked out.

By default, disabling the cursor works differently for window systems than for other devices. For window systems, the cursor is restored to the standard cursor used for non-PV-WAVE windows (and remains visible), while for other devices it is complete-ly blanked out.

Inches — If present, specifies that all position values are in inches from the origin.

Normal — If present and nonzero, specifies that the cursor position is in normalized coordinates.

T3D — If present, indicates that the generalized transformation matrix in !P.T is to be used. (For a description of !P.T, see the Save plotting keyword in Chapter 3, Graphics and Plotting Keywords.)

If not present, the user-supplied coordinates are simply scaled to screen coordinates.

Z — The Z position. The value of Z is of use only if the three-dimensional transformation is in effect via the T3D keyword.

Discussion

Normally, the cursor is disabled and is not visible. Using TVCRS with one parameter allows the cursor to be enabled or disabled, while using TVCRS with two parameters enables the cursor and places it on pixel location (X, Y).

Example

To enable the image display cursor and position it at device coordinate, use the following command:

```
TVCRS, 100, 100
```

To enable the image display cursor and position it at data coordinate, use the following command:

```
TVCRS, 0.5, 3.2, /Data
```

To disable and hide the image display cursor, use the following command:

```
TVCRS, /Hide Cursor
```

To enable the image display cursor but not set the cursor position, use the following command:

```
TVCRS, 1
```

See Also

TVRD, CURSOR

For more information, see Image Display Routines: TV and TVSCL on page 136 of the PV-WAVE User's Guide.

TVLCT Procedure

Loads the display color translation tables from the specified variables.

Usage

TVLCT, v_1 , v_2 , v_3 [, start]

Input Parameters

 v_1 , v_2 , v_3 — Contain the three values to be used for the specified color system (HLS, HSV, or RGB, as detailed in the Discussion section below). May be either scalar or vector expressions.

start - An integer value specifying the starting point in the color translation table into which the color intensities are loaded.

If start is not specified, a value of zero is used, causing the tables to be loaded starting at the first element of the translation tables.

Input Keywords

Hls – Indicates that the parameters specify color using the HLS color system.

Hsv - Indicates that the parameters specify color using the HSV color system.

Output Keywords

Get – If set to 1, returns the actual RGB values loaded into the device when either the Hls or Hsv keywords are present. (The Get keyword has no effect when loading RGB tables.)

For example, the statements:

```
TVLCT, H, S, V, /Hsv
TVLCT, R, G, B, /Get
```

load a color table based on the HSV system, and store the equivalent RGB values into the H, S, and V vectors.

Discussion

Color tables may be based on the following color systems: RGB (Red Green Blue, the default), HLS (Hue Lightness Saturation), and HSV (Hue Saturation Value).

The meaning and type for the v_1 , v_2 , and v_3 parameters are dependent upon the color system selected, as described below. If no color-system keywords are present, the RGB color system is used.

- HLS Parameters contain the Hue, Lightness, and Saturation values. All parameters are floating-point. Hue is expressed in degrees and is reduced modulo $360. v_2$ and v_3 may range from 0 to 1.0.
- HSV Parameters contain the Hue, Saturation, and Value (similar to intensity) values. All parameters are floating-point. Hue is expressed in degrees. The Saturation and Value may range from 0 to 1.0.
- RGB Parameters contain the Red, Green, and Blue values.
 Values are interpreted as integers in the range of 0 to 255 (255 being full intensity). May be scalars or may contain up to 256 elements.

See Also

LOADCT, MODIFYCT, COLOR_EDIT, STRETCH, TEK_COLOR, WgCtTool

For more information, including a comparison of TVLCT and LOADCT, see *Experimenting with Different Color Tables* on page 310 of the *PV*-WAVE User's Guide.

TVMENU Function

Standard Library procedure that creates an interactive menu.

Usage

result = TVMENU(text[, title, x, y])

Input Parameters

text — A string array with each element containing the text of one menu choice.

title - A string containing the title of the menu. The default is "Menu".

x — The X location of the lower-left corner of the menu, given in number of pixels from the lower-left corner of the display screen.

y - The Y location of the lower-left corner of the menu, given in number of pixels from the lower-left corner of the display screen.

Returned Value

result – The index of the choice. Values range from 0 (the first element in text) to one less than the number of elements in text.

Keywords

None.

Discussion

TVMENU is similar to the WMENU procedure. However, TVMENU creates a menu that can be positioned with the x and y parameters, and places it in a separate window.

This menu displays, in a column, the list of strings that were input as the text parameter. It waits while you use the mouse to click on the string of your choice. It then stores the value of that choice in the returned variable name. (Remember that PV-WAVE starts counting at 0, so your returned value will be one less than its position on the menu.)

Example

```
a = 5.
b = 7.
n = TVMENU(['add', 'subtract', 'multiply', $
    'divide'], 'Choose one:', 100, 100)
       Display the menu, and then make your choice by clicking
       with the mouse.
.RUN
- CASE n OF
- 0:PRINT, 'a+b=', a+b
- 1:PRINT, 'a-b=', a-b
- 2:PRINT, 'a*b=', a*b
- 3:PRINT, 'a/b=', a/b
- ENDCASE
- END
   Set up a case statement to perform the chosen function.
n = TVMENU(['add', 'subtract', 'multiply', $
   'divide'], 'Choose one:', 100, 100)
       Display the menu again.
.GO
   Rerun the saved program.
```

See Also

WMENU

TVRD Function

Returns the contents of the specified rectangular portion of a displayed image.

Usage

 $result = TVRD(x_0, y_0, n_x, n_y [, channel])$

Input Parameters

 x_0 - Starting column of data to read.

 y_0 - Starting row of data to read.

 n_r – Number of columns to read.

 n_v – Number of rows to read.

channel — The memory channel to be read. If not specified, it is assumed to be zero. This parameter is ignored on display systems that only have one memory channel.

Returned Value

result — Byte array of dimensions n_x -by- n_y .



If the display is a 24-bit display, and both the *channel* parameter and True keyword are absent, the maximum RGB value in each pixel is returned.

Input Keywords

Channel – The memory channel to be read. This keyword is identical to the channel input parameter; only one needs to be specified.

Order – If specified, overrides the current setting of the !Order system variable for the current image only. If nonzero, Order causes the image to be drawn from the top-down, instead of from the bottom-up (the default).

True — If present and nonzero, indicates that a true-color (24-bit) image is to be read and specifies the index of the dimension over which color is interleaved:

- 1 Displays an array that is pixel-interleaved and has dimensions of $(3, n_x, n_y)$.
- 2 Displays a line-interleaved array of dimensions $(n_x, 3, n_y)$.
- 3 Displays an image-interleaved array of dimensions $(n_x, n_y, 3)$.

Example

For an example of TVRD, see the REBIN function.

See Also

!Order, TV, TVSCL, ZOOM

For more information, see *Image Display Routines: TV and TVSCL* on page 136 of the *PV*-WAVE User's Guide.

TVSCL Procedure

Scales the intensity values of an input image into the range of the image display, usually from 0 to 255, and outputs the data to the image display at the specified location.

Usage

```
TVSCL, image [, x, y [, channel ]]
TVSCL, image [, position]
```

Input Parameters

image — A vector or two-dimensional matrix to be displayed as an image. If *image* is not already of byte type, it is converted prior to use.

x, y — The X and Y lower-left coordinates of the displayed image.

channel — The memory channel to be written. If not specified, it is assumed to be zero. This parameter is ignored on display systems that have only one memory channel.

position — Number specifying the position of the image. Positions run from the left of the window to the right, and from the top of the window to the bottom (see Discussion below for details).

Input Keywords

Centimeters — If present, specifies that all position values (the x y parameters and the XSize and YSize keywords) are in centimeters from the origin. This is useful when dealing with devices that do not provide a direct relationship between image pixels and the size of the resulting image, such as PostScript printers.

If Centimeters is not present, position values are taken to be in device coordinates.

Channel — The memory channel to be written. This keyword is identical to the channel input parameter; only one needs to be used.

Data — If present, specifies that all position and size values are in data coordinates. This is useful when drawing an image over an existing plot, since the plot establishes the data scaling.

Device — If present, specifies that all position and size values are in device coordinates. This is the default.

Inches — If present, specifies that all position and size values are in inches from the origin. This is useful when dealing with devices that do not provide a direct relationship between image pixels and the size of the resulting image, such as PostScript printers.

Normal — If present, specifies that all position and size values are in normalized coordinates in the range 0.0 to 1.0. This is useful when you want to draw an image in a device-independent manner.

Order — If specified, overrides the current setting of the !Order system variable for the current image only. If nonzero, *Order* causes the image to be drawn from the top-down, instead of from the bottom-up (the default).

True — If present and nonzero, indicates that a true-color image is to be displaced and specifies the index of the dimension over which color is interleaved:

- 1 Displays pixel-interleaved images of dimensions (3, m, n).
- 2 Displays row-interleaved images of dimensions (m, 3, n).
- 3 Displays image-interleaved images of dimensions (m, n, 3). (Image interleaving is also known as band interleaving.)



To use *True*, the *image* parameter must have three dimensions, one of which is equal to 3.

XSize — The width of the resulting image. This keyword is intended for use by devices with scalable pixel size (such as Post-Script), and is ignored by pixel-based devices that are unable to change the size of their pixels.

YSize — The height of the resulting image, with the same limitations as XSize.

Discussion

If position is used instead of the x and y parameters, the position of the image is calculated based on the largest grid of images of this size that will fit on the display, numbered from left to right and top to bottom. Specifically, the position is calculated as explained below.

The starting X coordinate position is defined as:

$$x = x_{dim} \cdot position_{moduloN}$$

The starting Y coordinate position is defined as:

$$y = YSize - y_{dim} \cdot int(1 + (position/N_x))$$

where

YSize is the height of the display or window,

 x_{dim} and y_{dim} are the dimensions of the array,

and the images across the display surface are defined as:

$$N_x = \frac{Xsize}{Ysize}$$

For example, when displaying 128-by-128 images on a 512-by-512 display, the position numbers run from 0 to 15 as follows:

Example

This example uses TVSCL and TV to exhibit the difference between images whose intensity values are scaled into the full range of the image display and images that have not been scaled.

```
OPENR, unit, FILEPATH('aerial_demo.img', $
Subdir='data'), /Get_Lun
Open the file containing the image.
```

img = BYTARR(512, 512)
Create an array large enough to hold the image.

READU, unit, img
Read the image data.

FREE_LUN, unit

Close the file and free the file unit number.

WINDOW, 0, Xsize = 1024, Ysize = 512

Create a window large enough to hold two 512-by-512 images.

TV, img

Display the original image in the left half of the window.

TVSCL, img, 1
Use TVSCL to display the scaled image in the right half of the window.

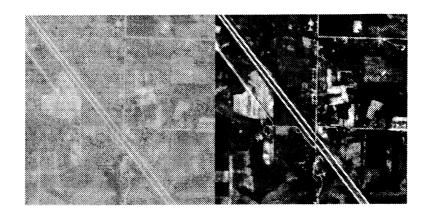


Figure 2-58 Unscaled image (left); scaled image (right).

See Also

!Order, BYTSCL, TVCRS, TVLCT, TVRD, TV

For more information, see Image Display Routines: TV and TVSCL on page 136 of the PV-WAVE User's Guide.

UNIQUE Function

Returns a vector (one-dimensional array) containing the unique elements from another vector.

Usage

```
result = UNIQUE(vec)
```

Input Parameters

vec – A vector containing duplicate values.

Returned Value

result — A new vector containing the unique elements of the original vector.

Keywords

None.

Discussion

This function works on any vector (one-dimensional array) variable.

Example 1

For a very simple example, suppose you have a vector defined as:

$$vec = [1,1,2,2,3,3,4,4,5,5]$$

The following command produces a new vector containing only the unique elements of vec:

```
result = UNIQUE(vec)
print, result
    1  2  3  4  5
```

Example 2

With UNIQUE, you can determine the unique values in any column of a table.

For example, a table called phone data contains information on phone calls made during a three-day period. This table contains eight columns of phone information: the date, time, caller's initials, phone extension, area code of call, number of call, duration, and cost. (For more information on the structure of this table, see Chapter 8, Creating and Querying Tables, in the PV-WAVE User's Guide.)

To obtain a list of the dates on which calls were made.

```
dates = UNIQUE(phone data.DATE)
```

Note that tables are represented internally in PV-WAVE as an array of structures. In this command, phone data. DATE represents the DATE field of the structure called phone data.

The result is a one-dimensional variable called dates that contains a list of the dates on which calls were made:

```
PRINT, dates
  901002 901003 901004
```

Example 3

In some applications, it may be possible to generate the text for menu buttons from the unique elements in a table column. For example, the following commands display a menu of dates. The menu selection is then passed into the QUERY TABLE function where the total cost of calls is calculated for that date.

```
unique date = UNIQUE(phone data.DATE)
    Find the unique dates in the phone data table.
date pick = unique date $
    (WMENU(STRING(unique date)))
        Display a menu of unique dates. The menu selection
        returns the selected date to the variable date pick. Note
        that the parameter passed to WMENU must be type string.
```

```
total_cost = QUERY_TABLE(phone_data, $
   'DATE, SUM(COST) Where DATE = ' +$
   'date_pick Group By DATE')
   Find the total cost of calls made on the selected date.
```

See Also

QUERY TABLE, BUILD TABLE

UNIX_LISTEN Function

Standard Library function that allows PV-WAVE to be called by external routines written in C.

Usage

result = UNIX LISTEN()

Parameters

None.

Returned Value

result — The number of parameters returned to UNIX LISTEN.

Input Keywords

Procedure — A string that is set by the client and retrieved by UNIX_LISTEN. Its intended use is to control program flow within the server (PV-WAVE).

User — A string that is set by the client and retrieved by UNIX_LISTEN. Its intended use is for controlling access to the server, as an authentication mechanism.

Program — An integer used as an identifier for the external routines. If an external routine (or client) calls call_wave, UNIX_LISTEN checks if the value of *Program* matches the value

of the program parameter in call_wave sent by the client. The client's program identifier is set by the program parameter in the call wave function call.

Discussion

UNIX LISTEN waits until an external routine calls the function call wave, and then returns the number of parameters passed to UNIX LISTEN.

Parameters are passed into PV-WAVE through the common block UT COMMON. A maximum of thirty parameters can be passed to PV-WAVE.

The common block UT COMMON is included in the server routine with the command @UT COMMON. The first of the thirty available parameters is ut param0, the second is ut_param1, and the thirtieth parameter is ut_param29.

As well as being returned by UNIX LISTEN, the number of parameters is also contained in the variable ut_num_params in the UT COMMON common block.

See Also

CALL UNIX, LINKNLOAD, SPAWN, UNIX REPLY

For more information and an example, see IntChapter 13, Interapplication Communication, in the PV-WAVE Programmer's Guide.

UNIX_REPLY Function

Standard Library function that allows PV-WAVE to return a value or values that it has calculated to an external routine written in C.

Usage

result = UNIX REPLY(reply)

Input Parameters

reply — A variable of any data type, except of type structure, representing the result of an operation that PV-WAVE, as a server, has performed.

Returned Value

result — A number indicating the status of the reply operation. A return value of -1 indicates an error. Errors can also be trapped by the ON IOERROR routine.

Output Keywords

Return_Params — If present and nonzero, causes UNIX_REPLY to return the modified parameters to the client. The number of parameters to be sent back is the same as the number that came in with UNIX_LISTEN. This number is tracked internally by PV-WAVE.

See Also

CALL_UNIX, LINKNLOAD, ON_IOERROR, SPAWN, UNIX_LISTEN

For more information and an example, see Chapter 13, *Interapplication Communication*, in the *PV*-WAVE Programmer's Guide.

USERSYM Procedure

Lets you create a custom symbol for marking plotted points.

Usage

USERSYM, x [, y]

Input Parameters

x, y – Vectors containing the X,Y vertices of the symbol to be created. x and y are offsets from the data point, in units of approximately the size of a character. In other words, a value of 1 corresponds to the actual X or Y size of a character, while a value of .5 is equal to one-half the character size.

In the case of vector-drawn symbols, these vertices are connected, in order, with vectors forming the symbol.

If only x is specified, it must be a (2, n) array of vertices, with element (0, i) containing the X coordinate of the vertex, and element (1, i) containing the Y coordinate.

Input Keywords

Color — An integer specifying the color used to draw the symbols, or used to fill the polygon. The default is the line color.

Fill - A flag which, if set, fills the polygon defined by the vertices. If Fill is not set, lines are drawn connecting the vertices.

Thick — The thickness of the lines drawn in constructing the symbol. The default is 1.0.

Discussion

Symbols may be drawn with vectors or may be filled. Symbols may be of any size and may have up to 50 vertices. To use a userdefined symbol, set the value of the *Psym* plotting keyword or the !Psym system variable to +8 or -8.

Example

This example uses USERSYM to define a plotting symbol that resembles a house. The *Fill* keyword is used to fill the interior of the polygon defining the house. Six random points are then plotted using the house symbol to mark the data points.

The *Psym* keyword, in the call to PLOT, is given a value of -8 so that the data points are connected by lines. Also, the *Symsize* keyword is used with a value of 6 to increase the size of the plotting symbols. This example uses PV-WAVE Advantage procedures RANDOMOPT and RANDOM.

```
x = [0, -0.5, -0.5, 0.5, 0.5, 0]
y = [0.5, 0, -1, -1, 0, 0.5]
   Define the x- and y-coordinates of the vertices.

USERSYM, x, y, /Fill
   Create the filled symbol.

RANDOMOPT, Set = 12542
   Generate six random points.

pts = RANDOM(6)

PLOT, pts, Psym = -8, Symsize = 6, $
   Xrange = [-0.1, 5.1]
   Plot the points, using the new user-defined symbol.
```

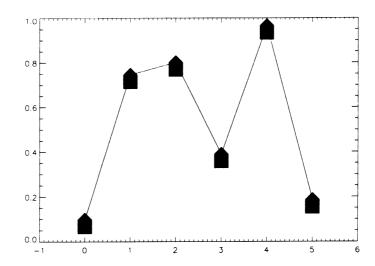


Figure 2-59 Scattered data plot with user-defined markers for data points.

See Also

!P.Psym system variable

Psym and Symsize plotting keywords

VAR_TO_DT Function

Converts scalar or array values representing dates and times to PV-WAVE Date/Time variables.

Usage

 $result = VAR_TO_DT (yyyy, mm, dd, hh, mn, ss)$

Input Parameters

yyyy - A scalar or array containing year number(s).

mm - A scalar or array containing month number(s) (1 - 12).

dd - A scalar or array containing day number(s) (1 - 31).

hh - A scalar or array containing hour number(s) (0-23). If zero or not specified, hh is 0 hours.

mn - A scalar or array containing minute number(s) (0 - 59). If zero or not specified, mn is 0 minutes.

ss - A scalar or array containing second number(s) (0.0000 - 59.9999). If zero or not specified, ss is 0.0 seconds.

Returned Value

result — A PV-WAVE Date/Time variable containing the converted values.

Keywords

None.

Discussion

Use this function to create Date/Time variables from time stamp data that does not conform to a format used by the STR_TO DT function. For example, you can read the date and time data into atomic PV-WAVE variables representing each of the date and time elements (i.e., year, month, day, etc.). Then the variables can be converted to Date/Time variables using VAR TO_DT.

If the year, month, and day values are all zero, then the value of the !DT Base system variable is used for the date portion of the resulting Date/Time variable.

The parameters can be arrays of any numeric values, but all parameters must have the same dimension; that is, the parameters must be either all scalars or all arrays of the same size. Also you can only omit parameters from the end of the parameter list.

Example 1

This example illustrates how to convert a single date value into PV-WAVE Date/Time variable.

```
z = VAR_{TO_DT(1992, 11, 22, 12, 30)}
   Note that seconds have been omitted. This command creates a
   Date/Time variable containing November 22, 1992 at 12:30.
PRINT, z
   { 1992 11 22 12 30 0.00000 87728.521 0}
DT PRINT, z
   11/22/1992
                  12:30:00
```

Example 2

This example illustrates how to return a PV-WAVE Date/Time variable for an array containing values representing dates.

```
years = [1992, 1993]
months = [3,4]
days = [17, 18]
```

Create arrays that contain Date/Time information for two days.

y = VAR_TO_DT(years, months, days)
Convert the Date/Time arrays to a Date/Time structure variable.

See Also

DT_TO_VAR, STR_TO_DT, SEC_TO DT, JUL TO DT

For more information, see the Chapter 7, Working with Date/Time Data, in the PV-WAVE User's Guide.

VECTOR FIELD3 Procedure

Plots a 3D vector field from three arrays.

Usage

VECTOR_FIELD3, vx, vy, vz, n points

Input Parameters

vx - A 3D array containing the X component of the vector field, or an *n*-element vector containing the X component of the vector field.

vy - A 3D array containing the Y component of the vector field, or an n-element vector containing the Y component of the vector field.

vz - A 3D array containing the Z component of the vector field, or an *n*-element vector containing the Z component of the vector field.



The arrays vx, vy, and vz must be the same size.

 n_points — If vx, vy, and vz are all 3D arrays and n_points is a (3, n) array, then n_points is used to specify where the vectors are plotted.

If vx, vy, and vz have the dimensions (i,j,k), then:

n points(0, *) should range between 0 and i - 1

n points(1,*) should range between 0 and j-1

n points(2, *) should range between 0 and k-1

If vx, vy, and vz are all 3D arrays and n points is a single positive value n, then n vectors are plotted with random starting locations.

If vx, vy, and vz are all 3D arrays and n points is zero or negative, then one vector is plotted for each element in vx.

If vx, vy, and vz are n-element vectors and n points is a (3, n) array, then the starting locations for each vector are taken from n_points .

Input Keywords

Axis_Color - The color to use when plotting the axis. To suppress the axis, set Axis Color to -1.

Mark Color – The color index to use when plotting the markers.

Mark Size — The marker size.

Mark Symbol – A number ranging from 1 to 7 defining the marker symbol to use. The default is 3 (a period). Markers are plotted at the tail of each vector. For a list of symbols, see the description of !Psym in Chapter 4, System Variables.

 Max_Color — The highest color index to use when plotting.

Min Color - The lowest color index to use when plotting. The color of each vector ranges from Min_Color to Max_Color.

 $Max \ Length - A$ scalar value for the maximum plotted length of each vector. Each vector ranges from zero to Max Length.

Thick — The line thickness (in pixels) to use when plotting vec-

 Vec_Color — A 3D array (with the same dimensions as vx) containing the data for the vector colors.

A vector plotted where *Vec_Color* is at its maximum has the color *Max_Color*, while a vector plotted at a location where *Vec_Color* is minimum has the color *Min_Color*.

If *Vec_Color* is not supplied, then the color of each vector is determined by its length.

Discussion

VECTOR_FIELD3 plots a 3D velocity vector field from volumetric or 3D data.

Examples

```
PRO vec demo1
   This program displays a 3D vector field from X, Y, Z data.
winx = 800
winy = 600
   Specify the window size.
v num = 1000
   Specify the number of vectors.
xvec = FLTARR(v num)
yvec = FLTARR(v num)
zvec = FLTARR(v num)
points = FLTARR(3, v_num)
   Create the arrays for the vectors and their starting points.
FOR k=0, 9 DO BEGIN
   FOR j=0, 9 DO BEGIN
      FOR i=0, 9 DO BEGIN
      ind = i + (j * 10) + (k * 10 * 10)
        xvec(ind) = COS(!PI * FLOAT(i)/10.0)
        yvec(ind) = SIN(!PI * FLOAT(j)/10.0)
         zvec(ind) = SIN(!PI * FLOAT(k)/10.0)$
           + COS(!PI * FLOAT(i)/10.0)
        points(*, ind) = [i, j, k]
      ENDFOR
```

ENDFOR

ENDFOR

Create the data for the vectors and their starting points.

```
T3D, /Reset
```

T3D, Translate=
$$[-0.5, -0.5, -0.5]$$

T3D, Scale=
$$[0.5, 0.5, 0.5]$$

T3D, Rotate=
$$[-60.0, 0.0, 0.0]$$

Set up the transformation matrix for the view.

LOADCT, 4 Set up the viewing window and load the color table.

```
VECTOR FIELD3, xvec, yvec, zvec, points, $
   Max Length=0.5, Min_Color=32, $
  Max Color=127, Axis_Color=100, $
  Mark Symbol=3, Mark_Color=127, $
  Mark Size=0.5, thick=2
```

Plot the vector field with the vector directions defined by xvec, yvec, and zvec, and the vector starting points defined by points.

END

For other examples, see the vec demo2 and vol demo1 demonstration programs in \$WAVE DIR/demo/arl.

See Also

VOL MARKER

VEL Procedure

Standard Library procedure that draws a graph of a velocity field with arrows pointing in the direction of the field. The length of an arrow is proportional to the strength of the field at that point.

Usage

VEL, u, v

Input Parameters

u — The X component of the velocity field at each point. Must be a two-dimensional array.

v — The Y component of the velocity field at each point. Must have the same dimensions as u.

Input Keywords

Nvecs — The number of arrows to draw. The default is 200.

Xmax — The aspect ratio (the X axis size as a fraction of the Y axis size). The default is 1.0.

Length — The length of each arrow segment, expressed as a fraction of the longest arrow divided by *Nsteps*. *Length* is used to calculate the proportional length of each arrow segment. The default is 0.1.

Nsteps — The number of segments in each arrow. The default is 10.

Discussion

VEL selects *Nvecs* random points within the boundary of the (u, v) arrays. At each point, the field is bilinearly interpolated from the (u, v) arrays and a vector of the correct proportional length is drawn.

The field is recalculated at the endpoint of this vector and a new vector is iteratively drawn, until an arrow with Nsteps number of segments is drawn.

An arrowhead is drawn at the end of this arrow, and the procedure moves on to another random point to initiate the loop again. The graph is plotted with the title "Velocity Field".

Caution

Extra care must be taken if you run the PLOT_FIELD and VEL procedures in the same PV-WAVE session. Each procedure calls a routine named ARROWS, but the ARROWS routines are slightly different. If you get an error in the ARROWS routine when you are using VEL, recompile VEL (by typing . RUN VEL), and then try again.

Example

```
u = FLTARR(21, 21)
v = FLTARR(21, 21)
   Create the arrays.
```

.RUN

After you type .RUN, the WAVE prompt changes to a dash (--) to indicate that you may enter a complete program unit.

```
- FOR j = 0, 20 DO BEGIN
- FOR i = 0, 20 DO BEGIN
-x = 0.05 * FLOAT(i)
-z = 0.05 * FLOAT(j)
-u(i, j) = -SIN(!Pi*x) * COS(!Pi*z)
-v(i, j) = COS(!Pi*x) * SIN(!Pi*z)
- ENDFOR
- ENDFOR
```

This procedure stuffs values into the arrays; the last END exits the programming mode, compiles and executes the procedure, and then returns you to the WAVE prompt.

```
VEL, u, v
```

- END

Display the velocity field with default values.

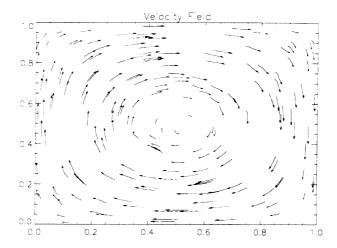


Figure 2-60 Velocity field displayed with default values.

VEL, u, v, Nvecs=400
Display the velocity field using 400 arrows.

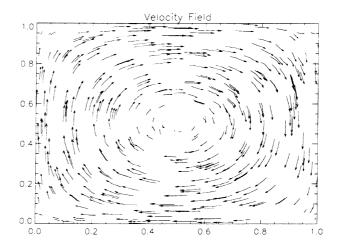


Figure 2-61 Velocity field displayed with 400 arrows.

VEL, u, v, Nvecs=40, Xmax=.7, Length=.4, \$ Nsteps=20

Display the velocity field with individual modifications.

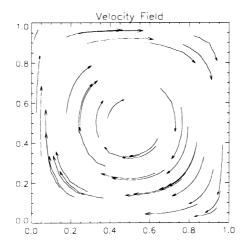


Figure 2-62 Velocity field displayed using various keywords.

See Also

PLOT_FIELD, VELOVECT

VELOVECT Procedure

Standard Library procedure that draws a two-dimensional velocity field plot, with each directed arrow indicating the magnitude and direction of the field.

Usage

VELOVECT, u, v [, x, y]

Input Parameters

- u The X component of the two-dimensional field. Must be a two-dimensional array.
- v The Y component of the two-dimensional field. Must be a two-dimensional array of the same size as u.
- x The abscissa values. Must be a vector. The size of x must equal the first dimension of u and v.
- y The ordinate values. Must be a vector. The size of y must equal the second dimension of u and v.

Input Keywords

Dots — If present and nonzero, places a dot at the position of the missing data. Otherwise, nothing is drawn for missing points. **Dots** is only valid if the *Missing* keyword is also specified.

Length — A length factor. The default value is 1.0, which makes the longest (u, v) vector have a length equal to the length of a single cell.

Missing — A two-dimensional array with the same size as the u and v arrays. It is used to specify that specific points have missing data.

If the magnitude of the vector at (i, j) is less than the corresponding value in Missing, then the data is considered to be valid. Otherwise, the data is considered to be missing.

Thus, one way to set up a Missing array is to initialize all elements to some large value:

```
missing array = FLTARR(n, m) + 1.0E30
```

Then, if point (i, j) is a missing point, set the corresponding element to a negative value:

```
missing_array(i, j) = -missing_array(i, j)
```

Discussion

VELOVECT draws a two-dimensional velocity field plot. The arrows indicate the magnitude and the direction of the field.

If missing values are present, you can use the Missing keyword to specify that they be ignored during the plotting, or the Dots keyword to specify that they be marked with a dot.

The system variables !X.Title, !Y.Title, and !P.Title may be used to title the axes and the main plot.

Example

```
u = FLTARR(21, 21)
v = FLTARR(21, 21)
   Create the arrays.
```

.RUN

- END

After you type .RUN, the WAVE prompt changes to a dash (-) to indicate that you may enter a complete program unit.

```
- FOR j = 0, 20 DO BEGIN
- FOR i = 0, 20 DO BEGIN
-x = 0.05 * FLOAT(i)
-z = 0.05 * FLOAT(j)
-u(i, j) = -SIN(!Pi*x) * COS(!Pi*z)
-v(i, j) = COS(!Pi*x) * SIN(!Pi*z)
- ENDFOR
- ENDFOR
```

This procedure stuffs values into the arrays; the last END exits the programming mode, compiles and executes the procedure. and then returns you to the WAVE prompt.

VELOVECT, u, v Display the velocity field with default values.

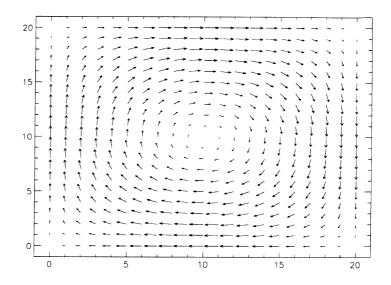


Figure 2-63 Velocity field displayed using default values.

VELOVECT, u, v, Length=2
Display the velocity field using arrows twice the length of a single cell.

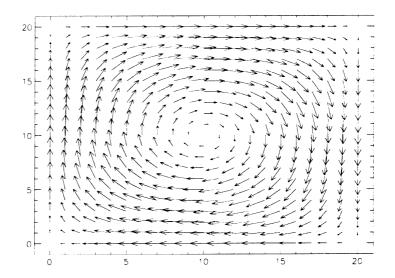


Figure 2-64 Velocity field displayed using arrows twice the length of a single cell.

```
missing = FLTARR(21, 21) + 1.e30
missing(4:6, 7:9) = -1.e30
missing(15:17, 16:19) = -1.e30
VELOVECT, u, v, Missing=missing, /Dots
   Display the velocity field that contains missing data.
```

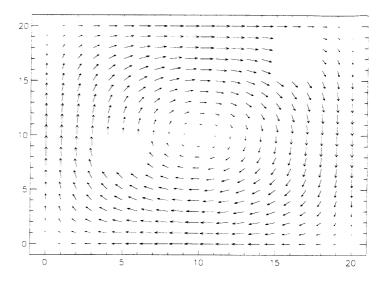


Figure 2-65 Velocity field that contains missing data.

See Also

PLOT_FIELD, VEL

VIEWER Procedure

Lets users interactively define a 3D view, a slicing plane, and multiple cut-away volumes for volume rendering. (Creates a View Control and a View Orientation window in which to make these definitions.)

Usage

VIEWER, win_num, xsize, ysize, size_fac, xpos, ypos, colors, retain, xdim, ydim, zdim

Input Parameters

win num — The number of the PV-WAVE window to use for the View Orientation window. Since this window is left on the screen when VIEWER returns, it is up to the calling program to delete it when desired. (The View Control window is automatically deleted when VIEWER returns.)

xsize, ysize — The X and Y dimensions, respectively, of the PV-WAVE window that VIEWER is setting up the view for.

size fac — A factor controlling the size of the View Orientation window. The X dimension of the View Orientation window is (size fac * xsize) and the Y dimension is (size fac * ysize). Typically, size fac should be in the range 0.5 to 1.0.

xpos, ypos — The location (X and Y positions, respectively) for the View Orientation and View Control windows.

If the View Orientation window is near the bottom of the screen. then the View Control window is created above the View Orientation window. Otherwise, the View Control window is created below the View Orientation window.

colors — The value (number of colors to allocate) to use as the Colors keyword value in the WINDOW procedure call. A typical value for an 8-bit color system is 128 or 256.

retain — The value (flag) to use as the **Retain** keyword value in the WINDOW procedure call. Typically this is the value 1 or 2.

For more information on *colors* and *retain*, see the description of the WINDOW procedure in the *PV*-WAVE Reference.

xdim, ydim, zdim — The size (first, second, and third dimension, respectively) of the array containing the data that is displayed using the view specified by VIEWER.

For example, if a 20-by-30-by-40 array is to be displayed using SHADE_VOLUME and POLYSHADE, then *xdim* should be 20, *ydim* should be 30, and *zdim* should be 40.

If a 20-by-30 array is to be displayed using the SURFACE or CONTOUR procedures, then *xdim* should be 20, *ydim* should be 30, and *zdim* should be (MAX(array)+1.0).

Input Keywords

Ax — On input, if this keyword is omitted, or if the variable passed to Ax is undefined, then no controls for setting the view rotation about the X axis are displayed.

If a valid value is passed to Ax, then this value is the initial (default) view rotation about the X axis.

Ay — The view rotation about the Y axis (similar to Ax).

Az — The view rotation about the Z axis (similar to Ax).

 $Bg\ Color$ — The window background color. The default is 0.

Bot_Color — The bottom shadow color for buttons. The default is 0.

Cut_Plane — On input, if this keyword is omitted, or if the variable passed to Cut_Plane is undefined, then no controls for setting the slicing plane are displayed.

For the cut-away controls to be displayed, a previously-defined variable must be passed to *Cut_Plane*. If this variable is a valid (3, 2) integer array, then it is assumed to contain the initial (default) slicing plane.

If cut plane is a valid array, then its contents are interpreted as follows:

cut_plane(0, 0)	The plane's angle of rotation about the X axis.
cut_plane(1, 0)	The plane's angle of rotation about the Y axis.
cut_plane(2, 0)	Ignored.
cut_plane(0, 1)	The X coordinate of the center of the plane.
cut_plane(1, 1)	The Y coordinate of the center of the plane.
cut_plane(2, 1)	The Z coordinate of the center of the plane.



The slicing plane is rotated about the Y axis first, then about the X axis. If the variable passed to Cut_Plane is defined but is not a valid (3, 2) array, then controls to set the slicing plane are displayed, the initial (default) slicing plane is defined with no rotation about the X or Y axes, and the center point of the plane is at (xdim/2, ydim/2, zdim/2).

 $Cut\ Vol$ — On input, if this keyword is omitted, or if the variable passed to Cut Vol is undefined, then no controls for setting the cutaway volume(s) are displayed.

For the cut-away controls to be displayed, a previously-defined variable must be passed to $Cut_Vol.$ If this variable is a valid (6, n)integer array, then it is assumed to contain the initial (default) cutting volumes. This (6, n) array contains the subscript ranges in the display array that are to be cut away.

For example, if a (6, 2) array called ca is passed in, then two initial cutting volumes are defined. In this case, the contents of ca are interpreted as follows:

ca(0, 0)	X dimension of the first cut-away.
ca(1, 0)	Y dimension of the first cut-away.
ca(2, 0)	Z dimension of the first cut-away.
ca(3, 0)	X position of the first cut-away.

ca(4, 0)	Y position of the first cut-away.
ca(5, 0)	Z position of the first cut-away.
ca(0:5, 1)	Defines second cut-away (similar to
	ca(0:5, 0)).

If the variable passed to Cut_Vol is defined, but is not a valid (6, n) array, then no initial cut-aways are defined, although the controls to define cut-aways are displayed.

Fg_Color — The foreground color. Used for buttons and for the current cut-away volume. The default is !P.Color.

HL_Color — The color to use when highlighting buttons and for drawing the slicing plane. The default is !P.Color.

Out_Mode — If present and nonzero, then users are not allowed to exit VIEWER until they have specified a view in which all the vertices of the view cube lie completely within the View Orientation window.



Setting *Out_Mode* to 1 ensures that VIEWER always sets up a view that is compatible with POLYSHADE. (POLYSHADE will not work properly if one or more polygon vertices lie outside the window.)

Persp — On input, if this keyword is omitted, or if the variable passed to **Persp** is undefined, then no controls for setting the view perspective projection distance are displayed.

If a valid value is passed to *Persp*, then this value is used as the initial (default) setting for the perspective projection, and controls for setting the perspective are displayed.

Top_Color — The top shadow color for buttons. Also used to draw the view cube. The default is !P.Color.

Zoom — On input, if this keyword is omitted, or if the variable passed to **Zoom** is not a valid scalar or three-element vector, then no controls for setting the **Zoom** factor(s) are displayed.

If the variable passed to *Zoom* is a single value, then this value is used as the initial (default) setting for the zoom factor and a single

control is provided for zooming the view equally in all three dimensions (X, Y, and Z).

If the variable passed to Zoom is a three-element vector, then these three values are the initial (default) values for the zoom factors and three controls are provided for zooming the view independently in the X, Y, and Z directions.

Output Keywords

Ax, Ay, Az — On output, these input keywords each return a single floating-point value containing the X, Y, and Z rotation, respectively, that was selected.

The calling program does not need to do anything with the returned values Ax, Ay, and Az, since VIEWER automatically calls CENTER VIEW to set the system view transformation !P.T, as well as !P.T3D, !X.S, !Y.S, and !Z.S. The calling program may, however, use these returned values in subsequent calls to VIEWER to let users "pick up where they left off."

Cut Plane — On output, the variable passed to Cut_Plane is always a valid (3, 2) single-precision floating-point array in the form described above under Input Keywords.

You can use the array returned from Cut Plane as input to the SLICE VOL procedure to extract the slice, or in subsequent calls to VIEWER.

Cut Vol - On output, the variable passed to Cut Vol is always a valid (6, n) integer array of the form described above under *Input* Keywords. If you do not define any cut-aways, then this variable is returned as a (6, 1) array containing all zeroes.

Typically, after the cut-away information is returned to the calling program, the array of data to be displayed is modified by the calling program using the cut-away information. This modification usually involves setting portions of the display array to zero before using the display array with other commands, such as RENDER, SHADE VOLUME, VOL REND, and VECTOR FIELD3.

For example, if the display array is a 20-by-30-by-40 array (xdim=20, ydim=30, zdim=40) and the variable returned from *Cut_Vol* is a (6, 1) array containing the values:

then the portions of the display array da to set to zero are as follows:

$$da(0:(0+19), 6:(6+10), 24:(24+13)) = 0$$

You may use the array returned from *Cut_Vol* in subsequent calls to VIEWER.

Persp — On output, this input keyword returns the perspective projection distance that was selected.

The calling program does not need to do anything with the returned values for *Zoom* or *Persp*, since VIEWER automatically sets the view. However, you may use the returned value(s) in subsequent calls to VIEWER.

Zoom — On output, this input keyword returns the **Zoom** value(s) that was selected.

Discussion

VIEWER returns the view parameters, slicing plane parameters, and cut-away volume coordinates to the calling program.

VIEWER automatically defines the view by calling the procedure CENTER_VIEW, but it is up to the calling program to decide what to do with any slicing plane or cut-away information returned to it. It is also up to the calling program to perform the rendering.



This procedure sets the system variables !P.T, !P.T3D, !X.S, !Y.S, and !Z.S, overriding any values you may have previously set. (These system variables are described in the *PV*-WAVE Reference.)

Due to the large amount of code in this procedure, the following command must be entered (once per PV-WAVE session) before any procedure that calls VIEWER can be run:

WAVE> .size 32766 32766

Interactive Usage

When VIEWER is called, a View Control and a View Orientation window are created that let users interactively define 3D viewing parameters. These windows are shown below in Figure 2-66 and Figure 2-67.

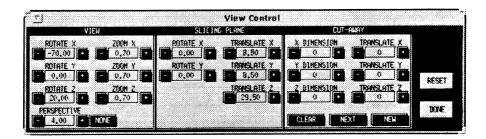


Figure 2-66 The View Control window contains buttons used for setting the view, slicing plane, and cut-away volume parameters.

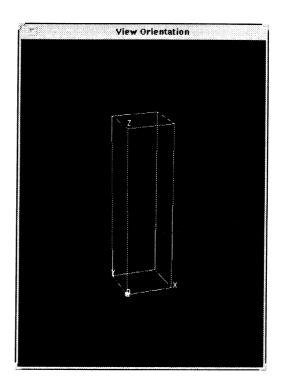


Figure 2-67 The View Orientation window displays a cube representing the current state of the parameters set in the View Control window.

The controls (buttons) provided for the user depend on the keywords supplied to VIEWER. For example, if you do not wish the user to be able to set the Y rotation parameter, then do not use the Ay keyword.

Note

It is advisable to use only the Ax, Az, and Zoom keywords when setting up a view for SURFACE or SHADE_SURF, since these procedures are limited in the type of viewing transformation they can utilize. (Other commands, such as PLOTS, POLYFILL, POLY_PLOT, and POLYSHADE, are compatible with most view transformations; therefore, you may freely use the Ay and Persp keywords when setting up the view for these routines.)

The available buttons in the View Control window are for X rotation, Y rotation, Z rotation, Perspective, and Zoom. Depending on how VIEWER is called, the Zoom keyword may consist of a single control, in which case the view is zoomed equally in the X, Y, and Z dimensions. There may also be three zoom controls provided, in which case the zoom factors for the X, Y, and Z dimensions may be set independently. There is also a NONE button provided for turning off the perspective projection.

Tip 📝

If the zoom factor is large, or the perspective parameter is small, then the cube display in the View Orientation window may be erroneous. To cure the problem, reduce the zoom or increase the perspective (or set the perspective to NONE).

All of the parameters can be changed by clicking on the plus [+] or minus [-] button for that parameter. The [+] button increases the value of that parameter and the [-] button decreases it. For rapid (coarse) control, use the left mouse button. For fine control, use the middle mouse button. For extra-fine control, use the right mouse button.

If the Cut Plane keyword is supplied to VIEWER, then controls are provided for setting the slicing plane. The rotation and position of the slicing plane can be set. The center of the slicing plane can not be located outside the original volume. The slicing plane is visible in the View Orientation window.

If the Cut Vol keyword is supplied to VIEWER, then controls are provided for setting the cut-away volume(s). The X size, Y size, Z size, X position, Y position, and Z position for each cut-away volume can be set. The size of the cut-away can not be larger than the original volume. No part of any cut-away volume may be positioned outside the original volume.

Three additional buttons, CLEAR, NEXT, and NEW, are also provided in the View Control window:

To create a new cut-away, click on the NEW button. This creates a small new cut-away near the coordinate origin. This cutaway is visible in the View Orientation window. You can then set the size and position of this cut-away by using the [+] and [-] buttons.

- If multiple cut-aways have been defined, the NEXT button allows users to toggle between them to modify their size and position.
- The CLEAR button removes all the cut-away volumes.

When you have finished setting the parameters, click the DONE button to set the view and return the parameters to the calling program.

If the *Out_Mode* keyword has been set, then clicking the DONE button does not cause a return to the calling program unless users have specified a view in which all the vertices of the view cube lie completely within the View Orientation window. In this case, users then need to change the view parameters (rotation, zoom, and/or perspective) before they could exit.

When VIEWER returns to the calling program, the View Control window is deleted, but the View Orientation window is left on the screen.

The RESET button can be used to reset all the parameters to their initial state.

Examples

For demonstrations of the VIEWER procedure, use the 4-D Data, Medical Imaging, Oil/Gas Exploration, and CFD/Aerospace buttons on the PV-WAVE Demonstration Gallery. To run the Gallery, enter wave gallery at the WAVE> prompt.

See Also

CENTER VIEW

VOL MARKER Procedure

Displays colored markers scattered throughout a volume.

Usage

VOL MARKER, vol, n points

Input Parameters

vol - A 3D volume of data to plot markers in.

n points — The number of markers to plot.

Input Keywords

Axis Color – The color to use when plotting the axis. To suppress the axis, set Axis Color to -1.

Mark Size - The maximum marker size.

Mark Symbol - A number specifying the marker symbol to use. The number should be between 1 and 7. The default is 2 (an asterisk). For a list of symbols, see the description of !Psym in Chapter 4, System Variables.

Mark Thick — The maximum line thickness to use when plotting markers. Typically, this is an integer between 1 and 7. (A thickness of 1.0 is normal, 2.0 is twice as wide, and so on.)

Discussion

VOL MARKER plots a polymarker field from volumetric data. The color of each marker displayed by VOL MARKER is determined by the value of the volumetric data at the point where the marker is plotted.

Examples

```
PRO vol_demo2
    This program displays an MRI scan of a human head using three
    different display techniques.
volx = 115
voly = 75
volz = 105
    Specify the size of the volumes.
winx = 512
winy = 512
    Specify the window size.
head = BYTARR(volx, voly, volz)
OPENR, 1, !Data_Dir + 'man head.dat'
READU, 1, head
CLOSE, 1
   Read in the volumetric data.
band = 5
head = VOL PAD(head, band)
head = SMOOTH(head, band)
   Pad the volume with zeroes and smooth it.
CENTER VIEW, Xr=[0, 124], Yr=[0, 84], $
   Zr=[0, 114], Az=60.0, Ax=(-60.0), $
   Winx=512, Winy=512, Zoom=0.9
WINDOW, 0, XSize=winx, YSize=winx, $
   XPos=16, YPos=384, Colors=128
LOADCT, 9
   Set up the viewing window and load the color table.
VOL MARKER, head, 6000, Axis Color=100, $
   Mark Symbol=2, Mark Size=2, Mark Thick=2
       Render the data using a 3D polymarker field.
WINDOW, 1, XSize=winx, YSize=winx, $
   XPos=496, YPos=324, Colors=128
       Create a second window for plotting.
```

- SET SHADING, Light=[-1.0, 1.0, 0.5], \$ /Gouraud, /Reject Change the direction of the light source for shading.
- SHADE VOLUME, head, 18, vertex_list, \$ polygon_list, /Low Compute the 3D contour surface as a list of polygons.
- TVSCL, POLYSHADE(vertex list, \$ polygon_list, XSize=winx, YSize=winy, \$ /Data, /T3d) Display the polygon list with light source shading.
- WINDOW, 2, XSize=winx, YSize=winx, \$ XPos=256, YPos=48, Colors=128 Create another window for plotting.
- head = VOL_TRANS(head, 128, !P.T) Transform the volumetric data to the current view.
- TVSCL, VOL_REND(head, winx, winy, Depth_Q=0.4) Display a translucent image of the data.

END

See Also

VECTOR_FIELD3

VOL_PAD Function

Returns a 3D volume of data padded on all six sides with zeroes.

Usage

result = VOL_PAD(volume, pad_width)

Input Parameters

volume — On input, volume contains the 3D volume of data to pad.

pad_width — The width of the padding around the volume. The size of the resulting volume increases by (2 * pad_width) in all three dimensions.

Returned Value

result — The padded volume data.

Keywords

None.

Discussion

For best results, process volumes with VOL_PAD before transforming them with VOL_TRANS or slicing them with SLICE_VOL. For more information, see *Volume Manipulation* on page 192 of the *PV-WAVE User's Guide*.

Examples

See the Examples section in the description of the VOL_MARKER routine.

For other examples, see the following demonstration programs in \$WAVE_DIR/demo/arl: vol_demo3, vol_demo4, and grid demo4.

VOL TRANS

VOL REND Function

Renders volumetric data in a translucent manner.

Usage

result = VOL REND(volume, imgx, imgy)

Input Parameters

volume — A 3D array containing volumetric data. volume is normally scaled into the range {0 ... 255}.

imgx – The X dimension of the image to return.

imgy — The Y dimension of the image to return.

Returned Value

result — An 8-bit image of the volumetric data.

Input Keywords

Depth q - A scalar depth queuing factor. **Depth** q should be between 0.0 and 1.0:

- A factor of 1.0 causes voxels in the back to be just as bright as the voxels in the front.
- A factor of 0.5 causes voxels in the back to be half as bright as those in front.

Opaque - A 3D array (with the same dimensions as volume) containing the translucency values for each voxel. Opaque is normally scaled into the range {0 ... 255}, where 0 is clear and 255 is completely opaque.

If *Opaque* is not specified, then the default is 0 (clear).

Discussion

If no keywords are used, the final intensity value produced at a given pixel with VOL_REND is the brightest value along the Z dimension of the volume. This default behavior can be enhanced, however, by using the *Opaque* and *Depth q* keywords.

- If *Opaque* is set to 0, the resulting value will be unaffected; if *Opaque* is set to 255, then values behind that position in the opaque array will be completely blocked.
- If Depth_q is set to a value less than 1.0, a bright spot in the back will be dimmed in proportion to the distance it is from the viewpoint. (A bright spot in the front will remain bright.)

Typically, you would first process a volume of data with VOL_PAD and VOL_TRANS, and then render it with VOL_REND.

Examples

```
PRO vol_demo3
   This program displays 3D fluid data using two display tech-
   niques.
volx = 17
voly = 17
volz = 59
   Specify the size of the volumes.
winx = 512
winy = 512
   Specify the window size.
flow axial = FLTARR(volx, voly, volz)
OPENR, 1, !Data Dir + 'cfd axial.dat', /Xdr
READU, 1, flow_axial
CLOSE, 1
flow_axial = VOL_PAD(flow_axial, 1)
   Read in the data and pad with zeroes.
```

```
CENTER VIEW, Xr = [0.0, 18.0], $
   Yr=[0.0, 18.0], Zr=[0.0, 60.0], $
   Az=210.0, Ay=120.0, Ax=0.0, Winx=512, $
   Winy=512, Zoom=0.85
       Set up the view.
```

SET SHADING, Light=[-1.0, 1.0, 0.5], \$ /Gouraud, /Reject Change the direction of the light source for shading.

SHADE VOLUME, flow axial, 110, vertex list, \$ polygon_list, /Low Compute the 3D contour surface as a list of polygons.

WINDOW, 1, XSize=winx, YSize=winx, XPos=16, \$ YPos=256, Colors=128

LOADCT, 3

Set up the viewing window and load the color table.

imq1 = POLYSHADE(vertex list, polygon list, \$ XSize=winx, YSize=winy, /Data, /T3d) TVSCL, img1

Construct the shaded surface representation of the data as a list of polygons and display it.

WINDOW, 2, XSize=winx, YSize=winx, \$ XPos=496, YPos=324, Colors=128 Create a new window for plotting.

vol_dim = MAX([volx, voly, volz]) flow axial = BYTSCL(flow axial) Scale the data into the range of bytes 0 - 255.

vol2 = VOL TRANS(flow axial, vol dim, !P.T) Transform the volume of data into the current view.

img2 = VOL REND(vol2, winx, winy, Depth q=0.4) Render the data as a translucent image.

TVSCL, img2 Display the image.

END

For other examples, see the following demonstration programs in \$WAVE_DIR/demo/arl: vol_demo2 and vol_demo4.

See Also

VOL_TRANS

VOL_TRANS Function

Returns a 3D volume of data transformed by a 4-by-4 matrix.

Usage

result = VOL TRANS(volume, dim, trans)

Input Parameters

volume - The 3D volume of data to transform.

dim — A scalar value specifying the X, Y, and Z dimensions of the transformed volume to return. Normally, dim is the largest of the three dimensions of the original volume. Generally, the original volume should "fit" inside the transformed volume.

trans — The 4-by-4 transformation matrix to use for the transformation. trans is often the system viewing transformation matrix !P.T. (For more information, see Geometric Transformations on page 167 of the PV-WAVE User's Guide.

Returned Value

result — A 3D volume of data transformed by a 4-by-4 matrix.

Keywords

None.

Discussion

The returned volume is scaled into the range of bytes. For best results, the volume to transform should first be processed using the VOL PAD function. For more information, see Volume Manipulation on page 192 of the PV-WAVE User's Guide.

Examples

See the Examples sections in the description of the VOL MARKER and VOL REND routines.

For another example, see the vol_demo4 demonstration program in \$WAVE DIR/demo/arl.

See Also

VOL_PAD, VOL_REND

VOLUME Function

Defines the volumetric data that can be used by the RENDER function.

Usage

result = VOLUME(voxels)

Input Parameters

voxels – A 3D byte array containing voxel data.

Returned Value

result — A structure that defines a volumetric object.

Input Keywords

Color — A 256-element double-precision floating-point vector containing the color (intensity) coefficients of the object. The default is Color(*)=1.0. For more information, see the section $Defining\ Color\ and\ Shading\ on\ page\ 198\ of\ the\ PV-WAVE\ User's\ Guide.$

Kamb — A 256-element double-precision floating-point vector containing the ambient (flat shaded) coefficients. The default is Kamb=FINDGEN(256)/255. For more information, see the section Ambient Component on page 199 of the PV-WAVE User's Guide.

Kdiff — A 256-element double-precision floating-point vector containing the diffuse reflectance coefficients. The default is Kdiff(*)=0.0. For more information, see the section *Diffuse Component* on page 198 of the *PV*-WAVE User's Guide.

Ktran — A 256-element double-precision floating-point vector containing the specular transmission coefficients. The default is Ktran(*)=0.0. For more information, see the section Transmission Component on page 199 of the PV-WAVE User's Guide.

Transform — A 4-by-4 double-precision floating-point array containing the local transformation matrix whose default is the identi-ty matrix. For more information, see the section Setting Object and View Transformations on page 201 of the PV-WAVE User's Guide.

Discussion

A VOLUME is used by the RENDER function to render volumetric data. You must specify a 3D array of bytes that represent this data in the call to VOLUME. Each byte in the voxel array corresponds to an index into the material properties associated with the volume.

For example, the material properties used for shading the point (x,y,z) in some data are Color (voxels (x,y,z)), Kamb(voxels(x,y,z)), etc. The surface normal at (x,y,z) is calculated using a 3D Sobel gradient operator on the actual voxel values. The default orientation of a volume is an origin-centered unit cube. For more information, see Defining Object Material Properties on page 200 of the PV-WAVE User's Guide.

If the voxels are not cubic, you may adjust the scaling of the dimensions with the Transform keyword by using a matrix generated by the T3D procedure with the Scale keyword.

Volumetric data is applicable to any voxel processing domain, such as for the visualization of astronomical, geological, and medical data.

Example

```
voxels = BYTARR(16, 16, 16)
voxels(*) = 255
diffuse = FLTARR(256)
T3D, /Reset, Rotate=[15, 30, 45]
cube = VOLUME(voxels, Transform=!P.T, $
   Kdiff=diffuse, Kamb=FLTARR(256))
TV, RENDER(cube)
```

See Also

CONE, CYLINDER, MESH, RENDER, SPHERE

For more information, see Ray-tracing Rendering on page 196 of the PV-WAVE User's Guide.

For details on the SHADE_VOLUME routine, see its description in the *PV*=*WAVE Reference*.

WAIT Procedure

Suspends execution of a PV-WAVE program for a specified period.

Usage

WAIT, seconds

Input Parameters

seconds — The duration of the wait, in seconds.

Keywords

None.

Discussion

WAIT is useful in interactive programs that repetitively read cursor positions.

Note that because of other activity on the system, the duration of program suspension may be longer than requested.

See Also

TVCRS, HAK, MESSAGE

WDELETE Procedure

Deletes the specified window.

Usage

WDELETE [, window index]

Input Parameters

window index — The window index of the window to be deleted. If not specified, deletes the current window, and the system variable !D. Window is set to the index of the first open window (or to -1 if no other windows are open).

Input Keywords

 X_No_Close — If present and nonzero, allows the window to be deleted for PV-WAVE, but remain active for the X Window System server. Otherwise, the window is deleted for both systems.

See Also

WINDOW, WSHOW, WSET, ERASE

WEOF Procedure

(VMS Only) Writes an end-of-file mark on the designated unit at the current position.

Usage

WEOF, unit

Input Parameters

unit — An integer between 0 and 9 specifying the magnetic tape unit on which the end-of-file mark will be written. (Do not confuse this parameter with file logical unit numbers.)

Keywords

None.

Discussion

To use WEOF, you must mount the tape as a foreign volume. The end-of-file mark is also sometimes called a tape mark.

See Also

TAPWRT, TAPRD, SKIPF

For more information, see Accessing Magnetic Tape on page 229 of the PV-WAVE Programmer's Guide.

WgAnimateTool Procedure

Creates a window for animating a sequence of images.

Usage

WgAnimateTool, image_data [, parent [, shell]]

Input Parameters

image data - A 3D array of images. The dimensions of the array are (m, n, n frames), where (m,n) is the size of an individual image, and n frames is the total number of frames in the sequence.

parent – (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgAnimateTool runs on its own (i.e., in its own event loop).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the AnimateTool window. Default value is "Animate Tool".

Cmap — The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the AnimateTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

Order – The order in which the image is drawn. If present and nonzero, the image is inverted. In other words, the image is drawn from bottom to top instead of from top to bottom.

Pixmap - 1 indicates that pixmaps should be used for the animation; 0 indicates that the data is stored in a PV-WAVE variable. Using pixmaps dramatically improves the speed of the animation, but requires more memory from the X Window System's X server than when data stored in variables is used.

 $\textbf{\textit{Do tvscl}} - 1$ indicates that TVSCL should be used to scale the image values to the current color table; 0 indicates that TV will be used instead (no scaling).

Delay – The minimum elapsed time between displayed images, specified in milliseconds (floating point).

File - A string containing the name of the file from which the image data is read. When using the File keyword, the Dims keyword must also be supplied. If present, the input variable image data is ignored.

Dims - A three-element vector specifying the size of the images to be read from the file, and the number of images to read. The elements of the vector are $[m, n, n_frames]$, where (m,n) is the size of an individual image, and n_frames is the total number of frames in the sequence.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

WgAnimateTool is an interactive window that lets you use the mouse to control the pace and direction of an animated series of images.



Using the WgAnimateTool window is similar in many ways to using the WgMovieTool window, but WgAnimateTool is intended to be used as a standalone utility widget, while WgMovieTool is designed so that it can be included inside larger layout widgets.

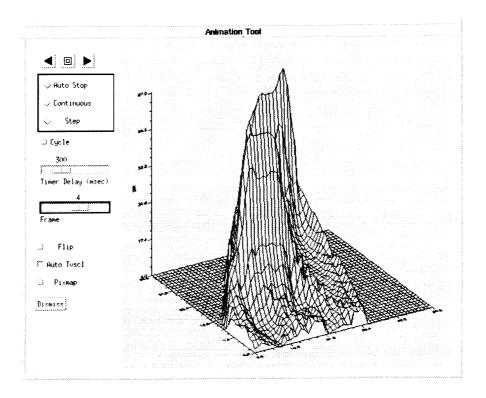


Figure 2-68 WgAnimateTool creates an interactive window that lets you use the mouse to control the orientation, pace, and direction of an animated series of images.

Contents of the Window

The AnimateTool window has two main parts — the display area and the control area.

AnimateTool Display Area

The display area is the largest area of the window; it is where the animation sequence takes place.

AnimateTool Control Area

Use the following controls to operate the AnimateTool window:

- **Reverse button** Cycle through the images from last to first.
- **Stop button** Freeze display at the current image.
- Forward button Cycle through the images from first to last.
- **Mode** Select Step to have the sequence proceed only when you click on one of the arrow buttons. Select Continuous to have AnimateTool move through the images in a non-stop manner, pausing only when you click the Stop button. Select Auto Stop to have AnimateTool stop sequencing when it reaches either end of the data.
- Cycle When enabled, the animation sequences repeatedly through the data, alternating between forward and reverse. When disabled, the animation returns to the first (or last) image in image data after every image in the sequence has been displayed.
- **Timer delay** Decrease or increase the rate of display. The number shown above the slider is the number of milliseconds delay between contiguous images in the animation sequence.
- Frame The ordinal number of the currently displayed image is displayed above this slider. If you wish, use the left mouse button to drag the slider and display a different image.
- Flip When enabled, the first data value is used to draw the pixel in the upper-left corner of the image. When disabled, the first data value is used to draw the pixel in the lower-left corner of the image.

- **Auto Tvscl** By determining whether the TV or TVSCL is used to draw the images, controls whether the input image data is automatically scaled to use the full range of available colors. Selecting this option may increase the contrast of the displayed images. Auto Tvscl does not affect the actual data; it only affects the display of the data.
- **Pixmap** When enabled, the data is stored in pixmaps; when disabled, the data is taken directly from the PV-WAVE variable.
- **Dismiss** Destroy the AnimateTool window and erase it from the screen.

Input Data Requirements

The animation can use data from either pixmaps or a PV-WAVE variable. Although the animation will run faster using pixmaps, it does require more memory to store the data as a pixmap than as a variable.

When reading the data from a file, the file containing the data must be a binary file containing the images in sequential order.

Event Handling

You can use the AnimateTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the AnimateTool widget. The AnimateTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another application — The AnimateTool widget has its own Main window, but the application (not the AnimateTool widget) handles the event loop by calling WwLoop.

The output parameter shell can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgAnimateTool is created as a child of parent; otherwise, WgAnimateTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgAnimateTool window, close it by clicking the Dismiss button.

```
PRO Sample_wganimatetool, parent, tool shell
heart = BYTARR(256, 256, 15)
    Define a variable to hold the data.
IF !Version.platform EQ 'VMS' THEN $
    OPENR, u, GETENV('WAVE DIR')+$
    '[data]heartbeat.dat', /Get_lun $
ENDIF ELSE BEGIN
OPENR, u, !Dir+'/data/heartbeat.dat', $
    /Get lun
ENDELSE
READU, u, heart
    Read the PV=WAVE heartbeat.dat demo file that contains
    images showing a beating human heart.
CLOSE, u
FREE LUN, u
    Close the file and free the LUN.
IF N ELEMENTS(parent) NE O THEN BEGIN
WgAnimateTool, heart, parent, tool shell, $
    /Do tvscl, /Pixmap
    Create WgAnimateTool as a child of the widget known as "par-
    ent". The window of the newly created widget is returned via the
   optional output parameter "tool shell".
ENDIF ELSE BEGIN
WgAnimateTool, heart, /Do tvscl, /Pixmap
   Create WgAnimateTool and display it as its own Main window.
   In other words, the WgAnimateTool window runs on its own (i.e.,
   in its own event loop).
```

ENDELSE END

See Also

MOVIE, WgMovieTool

For more information about how the UNIX environment variable \$WAVE GUI or the VMS logical WAVE GUI affects whether the AnimateTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about how to transfer image data to PV-WAVE variables, refer to Input and Output of Image Data on page 189 of the PV-WAVE Programmer's Guide.

For more information about pixmaps, refer to *Using Pixmaps to* Improve Application Performance on page A-85 of the PV-WAVE User's Guide.

For more information about color table indices, refer to Experimenting with Different Color Tables on page 310 of the PV-WAVE User's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide. For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgCbarTool Procedure

Creates a simple color bar that can be used to view and interactively shift a PV-WAVE color table.

Usage

WgCbarTool [, parent [, shell [, windowid [, movedCallback], [, range]]]]]

Input Parameters

parent — (optional) The widget or shell ID of the parent widget (long). If **parent** is not specified, WgCbarTool runs on its own (i.e., in its own event loop).

movedCallback — (optional) A string containing the name of the PV-WAVE callback routine that is executed when the color bar is shifted to the left or right.

range – (optional) The range of colors to be displayed in the color bar. The default is to display the entire range of colors, as defined in the system variable !D.Table_Size.

Output Parameters

shell — (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

windowid — (optional) The window ID of the PV-WAVE graphics window. (For information on window IDs, see the description for the WINDOW procedure.)

Input Keywords

Title — A string containing the title that appears in the header of the CbarTool window. Default value is "Color Bar".

Cmap — The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the CbarTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are either: 1) if parent is present and the Popup keyword is not specified or has a value of zero, measured from the upper-left corner of a layout (container) widget or 2) if the *Popup* keyword is present and nonzero (regardless of whether parent is present and/or nonzero), measured from the upper-left corner of the screen.

Popup — If present and nonzero, the color bar widget is displayed in its own Main window.

Size - A two-element vector specifying the width and height of the color bar (long integer). If not specified, the default size of the color bar is 30-by-256 pixels.

Vertical — If present and nonzero, the color bar is oriented vertically (default).

Horizontal — If present and nonzero, the color bar is oriented horizontally.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font — Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom=wid), then the bottom of the color bar widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the color bar widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the color bar widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the color bar widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the color bar widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the color bar widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the color bar widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the color bar widget is attached to the top of the parent widget.

Discussion

The color bar lets you display the colors of the current color table and "rotate" them.

To rotate the color table using the color bar, press and drag the left mouse button inside the color bar. As you "slide" colors into different color table indices, the colors that "scroll off" the end of the color table are added to the opposite end.



Use the window manager menu of the window frame to dismiss the CbarTool window from the screen.



Figure 2-69 WgCbarTool creates an array of colors that match the colors in the current colortable. This color bar widget has been created using the /Horizontal option; the default is for the color bar to be displayed in a vertical orientation.

Event Handling

You can use the color bar widget in one of the following three ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the color bar widget. The color bar widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another **application** – If the *Popup* keyword is present and nonzero, the color bar widget has its own Main window. The application (not the color bar widget) handles the event loop by calling WwLoop.
- Combined with other widgets in a layout widget Another application combines the color bar widget with other widgets inside a layout widget. The application (not the color bar widget) handles the event loop by calling WwLoop.

When the input parameter parent is present and nonzero, the widget operates in either the second or third mode listed, depending on the value of the Popup keyword. The output parameter shell can be returned only if you also supply the input parameter parent.

You can use the output parameter windowid to keep track of the PV-WAVE window ID that is assigned to the color bar. You can create multiple instances of the color bar; each one will be assigned a different PV-WAVE window ID.

The Colors Common Block

This procedure modifies the PV-WAVE internal color table, as well as the color table variables in the Colors common block.

Most color table procedures maintain the current color table in a common block called Colors, defined as follows:

```
COMMON Colors, r_orig, g_orig, b_orig, $
   r_curr, g_curr, b_curr
```

The variables are integer vectors of length equal to the number of color indices. Your program can access these variables by declaring the common block. The modifications you make to the color table by interacting with the CbarTool widget are stored in r_curr, g_curr, and b curr.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgCbarTool is created as a child of parent; otherwise, WgCbarTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgCbarTool window, close it using the window manager menu.

```
PRO Sample_wgcbartool, parent, tool_shell

IF N_ELEMENTS(parent) NE 0 THEN BEGIN

WgCbarTool, parent, tool_shell, /Horizontal, $
    /Popup
```

Create WgCbarTool as a child of the widget known as "parent". The window of the newly created widget is returned via the optional output parameter "tool_shell".

```
ENDIF ELSE BEGIN
```

WgCbarTool, /Horizontal

Create WgCbarTool and display it as its own Main window. In other words, the WgCbarTool window runs on its own (i.e., in its own event loop).

ENDELSE

END

See Also

WgCeditTool

For more information about how the UNIX environment variable \$WAVE GUI or the VMS logical WAVE GUI affects whether the CbarTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about color table indices, refer to Experimenting with Different Color Tables on page 310 of the PV-WAVE User's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide. For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgCeditTool Procedure

Creates a full-featured set of menus and widgets enclosed in a window; this window allows you to edit the values in PV-WAVE color tables in many different ways.

Usage

WgCeditTool [, parent [, shell]]

Input Parameters

parent — (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgCeditTool runs on its own (i.e., in its own event loop).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Cmap — The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the CeditTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

Title — A string containing the title that appears in the header of the CeditTool window. Default value is "Color Editor".

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor - Specifies the base color.

Font — Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

Using the features of this window, you can edit individual color table values, or you can change ranges of values. (Range of colors can be selected and linearly interpolated.) You can use either sliders or a color wheel to adjust the current color.

You use a palette of color cells to select the current color - point to the color that interests you and click the left mouse button to select it. In this sense, the CeditTool window is like an artist's palette, where basic colors are mixed to produce different colors.

You can use either the RGB, HLS, or the HSV color system to create a new color table, and you can view the result as a color bar or as a set of three intensity graphs. You can store color tables you have modified as custom color tables, and they will be available for your later use.

In some ways, the WgCeditTool window is similar to the WgCtTool window, in that it provides easy ways to interactively modify color tables. For example, with just a few clicks, you can use the WgCeditTool window to edit individual colors in the color table. Or you can use the Options menu to open several other utility widgets. But if you do not need all the options of the WgCeditTool window, or if you just need a quick way to stretch or reverse a color table, then use the WgCtTool window, instead.

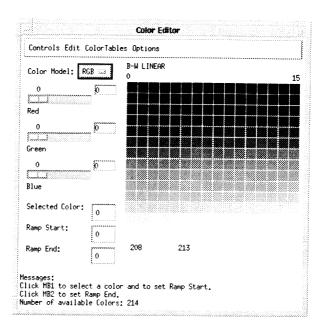


Figure 2-70 The WgCeditTool window lets you use the mouse to create a new color table based on either the HLS, HSV, or RGB color systems.

What is a Color Table?

On workstations that support color, the specific colors that are used depend on the current *color table*. A color table maps data values into colors. Many workstations support thousands of different colors, but only some number, usually 256, can be displayed at any one time.

By default, WgCeditTool uses as many colors in the color table as are currently available in the workstation's colormap. This number depends on the colors that have already been allocated by other applications running on that workstation.

The Colors Common Block

This procedure modifies the PV-WAVE internal color table, as well as the color table variables in the COLORS common block.

Most color table procedures maintain the current color table in a common block called Colors, defined as follows:

```
COMMON Colors, r orig, g orig, b_orig, $
   r curr, g curr, b curr
```

The variables are integer vectors of length equal to the number of color indices. Your program can access these variables by declaring the common block. The modifications you make to the color table by interacting with the CeditTool widget are stored in r curr, g curr, and b curr.

Custom Color Table File

The names of custom color tables that have been saved are stored in a file named .wg colors. This file is placed in your home directory, which in UNIX is defined by \$HOME, and in VMS is defined by SYS\$LOGIN.



Do not attempt to edit the .wg colors file — doing so could lead to unpredictable results.

Contents of the Window

The CeditTool window has three main parts — the color palette area, the control area, and the message area. The options provided by the menu bar are discussed in a later section.

CeditTool Color Palette Area

An array of cells, one for each color index from the colormap. A message near the bottom of the window informs you how many colors were available to PV-WAVE, and thus how many color cells could be displayed.

CeditTool Control Area

The window's control area contains sliders, a menu, and text fields that are used to manipulate colors:

- Color Model (menu) Choose a color system from the menu. By default, the CeditTool uses the RGB color system. For more information about color systems, refer to *Understanding Color Systems* on page 305 of the *PV-WAVE User's Guide*.
- Color Cell Text Fields and Sliders These controls allow you to modify the three basic components of any color in the color table. The labels on the text fields and sliders change to coincide with the current color system. First, select a color to modify by clicking on one of the color cells in the display area, or by entering its index number in the Selected Color text field. To modify the selected color, either enter a new color value in one of the text fields, or use the slider to change the value. If you enter a new color value into a text field, press <Return> to apply the new value to the slider and the color cell that corresponds to the selected color. If you use the sliders, the change is applied immediately as the slider moves.
- Selected Color The index number of the currently selected color in the color table. Once a color is selected, other sliders and text fields in the tool control area can be used to modify its composition. Select a color by clicking the left mouse button on a color cell in the palette of cells, or by entering the color's index number in this text field and pressing <Return>.
- Ramp Start and End Two text fields where you can enter color table indices between which the CeditTool will perform a linear color interpolation; this is an easy way to edit the current color table. Either enter the beginning and ending color indices in the Ramp Start and Ramp End text fields or use the mouse to select the ramp's starting and ending colors. (To use the mouse, click the left mouse button on the starting color cell and the middle mouse button on the ending color cell.) Select Edit=>Ramp on the CeditTool window. The affected cells in

the displayed color table are updated immediately, although no permanent change is made until you save the color table as a custom color table.

CeditTool Menu Bar

The CeditTool menu bar consists of four menu buttons located near the top of the window:

Controls Menu

- Save Custom Save the current set of color indices using the same custom color table name as before.
- Save As Custom Save the current set of color indices as a new custom color table with a unique name.
- **Rename Custom** Rename a custom color table. You must select the custom color table from the Custom Color Table option menu (select ColorTables=>Custom) before you can rename it.
- **Delete Custom** Delete one of the custom color tables from the list. For details about where this list of color tables is stored, refer to Custom Color Table File on page 293.
- **Exit** Destroy the CeditTool window and all its children.

Edit Menu

- **Ramp** Create a color ramp between the start ramp color and the end ramp color. See the description of Ramp Start and Ramp End on the previous page for more information about choosing start and end ramp colors.
- **Restore** Restore the original color map for the selected color table.

ColorTables Menu

- System Display a list of system color tables. System color tables are provided with PV-WAVE and cannot be modified.
- **Custom** Display a list of custom color tables. These are color tables that you have created using Controls=>Save Custom or Controls=>Save As Custom (from the Controls menu).

Options Menu

Color Wheel – Select the HLS or HSV components for a particular color by interacting with a dial and a slider; the color wheel is shown in Figure 2-71.

The characteristics you can adjust in the dial are: 1) hue (azimuth of mouse from the center), and 2) saturation (distance from the center).

The characteristics you can adjust in the vertical slider are value (in the HSV color system) or lightness (in the HLS color system).

Tip

The farther you click from the center of the wheel (with the left mouse button), the more saturated the color (for that particular hue).

- **Intensity Graphs** The value of each of the three color system parameters (HLS or HSV) is plotted versus pixel value in a separate line graph. This results in three line graphs showing the current values of the three parameters for the current color table, as shown in Figure 2-71.
- **Color Bar** Display a vertical color bar; this color bar displays every one of the colors in the current color table; a fullrange color bar is shown in Figure 2-71.

The color bar feature lets you display the colors of the current color table and "rotate" them. To rotate the color table using the color bar, press and drag the left mouse button up or down in the color bar.

If you have any other CeditTool options displayed on your screen, like the intensity graphs or the color wheel, you will see immediate effects as the color table shifts in relation to the mouse's movement.

Note

As you "slide" colors into different color table indices, the colors that "scroll off" the end of the color table are added to the opposite end.

Color Bar Range — Display a vertical color bar displaying only the range of selected colors; a partial-range color bar is shown in Figure 2-71. The range of selected colors is the same as the range chosen for the linear interpolation in the text fields Ramp Start and Ramp End.

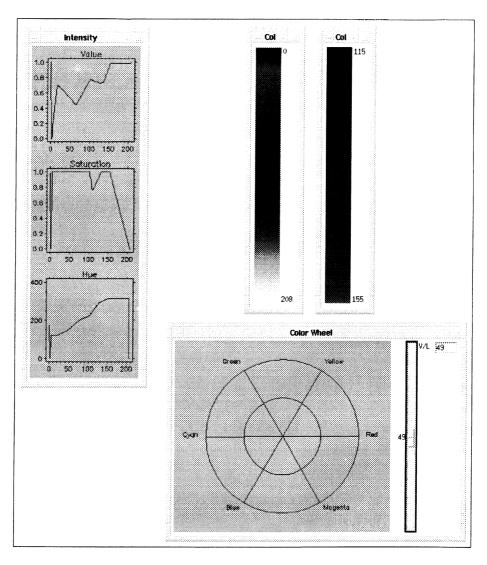


Figure 2-71 Utility widgets accessible via WgCeditTool's Options menu. The four utility widgets are (starting from the upper left, proceeding clockwise): 1) intensity graphs, 2) a color bar showing the entire range of the color table, 3) a color bar showing only a selected range of the color table (range modifiable by user), and 4) a color wheel and slider that can be used to interactively modify the current color. For more information about any of these utility widgets, refer to the Options Menu section.

Event Handling

You can use the CeditTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the CeditTool widget. The CeditTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another application — The CeditTool widget has its own Main window, but the application (not the CeditTool widget) handles the event loop by calling WwLoop.

The output parameter shell can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgCeditTool is created as a child of parent; otherwise, WgCeditTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgCeditTool window, close it by selecting Controls=>Exit from the menus of the Cedit-Tool window.

```
PRO Sample wgcedittool, parent, tool_shell
IF N ELEMENTS(parent) NE 0 THEN BEGIN
WqCeditTool, parent, tool shell
   Create WgCeditTool as a child of the widget known as "parent".
   The window of the newly created widget is returned via the
   optional output parameter "tool shell".
```

```
ENDIF ELSE BEGIN
```

WqCeditTool

Create WgCeditTool and display it as its own Main window. In other words, the WgCeditTool window runs on its own (i.e., in its own event loop).

ENDELSE

END

See Also

WgCtTool, COLOR_PALETTE, COLOR_EDIT

For more information about how the UNIX environment variable \$WAVE_GUI or the VMS logical WAVE_GUI affects whether the CeditTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about color table indices, refer to Experimenting with Different Color Tables on page 310 of the PV-WAVE User's Guide.

For more information about color systems, refer to *Understanding Color Systems* on page 305 of the *PV*-WAVE User's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*. For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WgCtTool Procedure

Creates a simple widget that can be used interactively to modify a PV-WAVE color table.

Usage

WgCtTool [, parent [, shell]]

Input Parameters

parent – (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgCtTool runs on its own (i.e., in its own event loop).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the window. Default value is "Color Table Tool".

Cmap — The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the CtTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor – Specifies the base color.

Font — Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

The WgCtTool window allows you to interactively modify system color tables by stretching, rotating, and reversing them.

In some ways, the WgCtTool window is similar to the WgCedit-Tool window, in that it provides easy ways to interactively modify color tables. For example, with a single click, you can reverse the color table using the WgCtTool window. But if you need to edit individual colors in the color table, or save your changes for future use, then use the WgCeditTool window, instead.

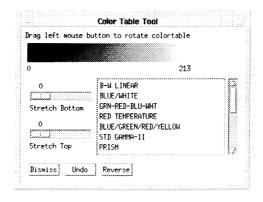


Figure 2-72 The WgCtTool window lets you interactively modify system color tables by stretching, rotating, and reversing them.

The Colors Common Block

This procedure modifies the PV-WAVE internal color table, as well as the color table variables in the COLORS common block.

Most color table procedures maintain the current color table in a common block called Colors, defined as follows:

```
COMMON Colors, r orig, g orig, b orig, $
   r_curr, g_curr, b_curr
```

The variables are integer vectors of length equal to the number of color indices. Your program can access these variables by declaring the common block. The modifications you make to the color table by interacting with the CtTool widget are stored in r curr, g curr, and b curr.

Contents of the Window

The CtTool window has three main parts — the color bar area, the control area, and the system color table list.

CtTool Color Bar Area

The built-in color bar lets you display the colors of the current color table and "rotate" them. To rotate the color table using the color bar, press and drag the left mouse button to the left or to the right in the color bar. As you "slide" colors into different color table indices, the colors that "scroll off" the end of the color table are added to the opposite end.

CtTool Control Area

Use the following controls to operate the CtTool window:

- Stretch Bottom Select the lower limit for the stretched color table.
- **Stretch Top** Select the upper limit for the stretched color table.

- Undo Discard color table modifications you have made using the CtTool window, and return to the color table that CtTool was using when it was first created.
- Reverse Swap the color table (as currently defined), end for end.
- **Dismiss** Destroy the CtTool window and erase it from the screen.

The CtTool window uses the STRETCH procedure to linearly expand a range of color table indices. The Stretch Bottom number is used for the first parameter to the STRETCH command, and the Stretch Top number is subtracted from the number of colors available in the color table to determine the second parameter to the STRETCH command. For more information about the STRETCH command, refer the description for STRETCH in the *PV-WAVE Reference*.

System Color Table List

This area lists the 16 system color tables that are provided as part of PV-WAVE. Use the left mouse button to select one. The color table you select from the list will immediately be displayed in the color bar area.

Because system color tables are "read-only", no system color table will be permanently altered by any changes you make with the CtTool window. For this reason, the changes you make with CtTool are temporary and can be overwritten by any other PV-WAVE routine writing to the Colors common block.



To save color table changes for later use, you can use another utility widget, WgCeditTool.

Event Handling

You can use the CtTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the CtTool widget. The CtTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another **application** — The CtTool widget has its own Main window, but the application (not the CtTool widget) handles the event loop by calling WwLoop.

The output parameter shell can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgCtTool is created as a child of parent; otherwise, WgCtTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgCtTool window, close it by clicking the Dismiss button.

```
PRO Sample_wgcttool, parent, tool_shell
IF N ELEMENTS(parent) NE O THEN BEGIN
WgCtTool, parent, tool shell
```

Create WgCtTool as a child of the widget known as "parent". The window of the newly created widget is returned via the optional output parameter "tool_shell".

```
ENDIF ELSE BEGIN
```

WgCtTool

Create WgCtTool and display it as its own Main window. In other words, the WgCtTool window runs on its own (i.e., in its own event loop).

ENDELSE

END

See Also

WgCeditTool, COLOR_PALETTE, COLOR_EDIT

For more information about how the UNIX environment variable \$WAVE_GUI or the VMS logical WAVE_GUI affects whether the CtTool window is implemented using the Motif or the OLIT lookand-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about color table indices, refer to Experimenting with Different Color Tables on page 310 of the PV-WAVE User's Guide.

For more information about color systems, refer to *Understanding Color Systems* on page 305 of the *PV*-WAVE User's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*. For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WglsoSurfTool Procedure

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.

Usage

WgIsoSurfTool, surface data [, parent [, shell]]

Input Parameters

surface_data - A 3D variable containing volumetric surface data. The dimensions of *surface data* define the size of the cube that you see when the IsoSurfTool window is first opened.

parent – (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgIsoSurfTool runs on its own (i.e., in its own event loop).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the IsoSurfTool window. Default value is "Isosurface Tool".

Cmap — The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Cube color — The color in which the rotatable cube is drawn; a positive integer in the range $\{0...255\}$. The number that is provided for cube color is used as an index into the current color table.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the IsoSurfTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

View rot — A three-element vector specifying the initial view rotation, in degrees. If not supplied, the initial rotation of the cube and the iso-surface is [30, 30, 0] — in other words, 30 degrees rotation around the X axis, 30 degrees rotation around the Y axis, and 0 degrees rotation around the Z axis.

View zoom — The initial zoom factor; default value is 0.5.

View_persp — The initial perspective distance. If *View_persp* is zero, then the initial setting on the Perspective Distance slider is 0.5, but the Perspective pushbutton is initially deselected. If View persp is nonzero, then the initial setting on the Perspective Distance slider is the value specified by View persp and the Perspective pushbutton is selected.

Note //

Using no perspective to draw the cube is equivalent to having the eyepoint an infinite distance away from the cube, and will produce a cube with sides that all appear to be parallel to one another.

Thresh range – The range of values to display in the threshold slider. The default is to use the entire range of values defined by surface_data, or in other words, the range:

{min surface data...max surface data}

Thresh value — The initial position of the threshold slider. By default, the movable portion of the slider is positioned in the middle of the threshold range.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font — Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

WgIsoSurfTool is an interactive window that lets you use the mouse to view and modify iso-surfaces; you can view any isosurface contained within the three-dimensional input dataset, surface data.

If the zoom factor is large, or the perspective parameter is small, then the cube display in the View Orientation window may be erroneous. To cure the problem, reduce the zoom or increase the perspective (or disable the perspective entirely).



Some iso-surfaces can be time-consuming to render, especially those that compose many polygons. PV-WAVE will display its "wait cursor" while it is performing the calculations necessary to display the iso-surface.

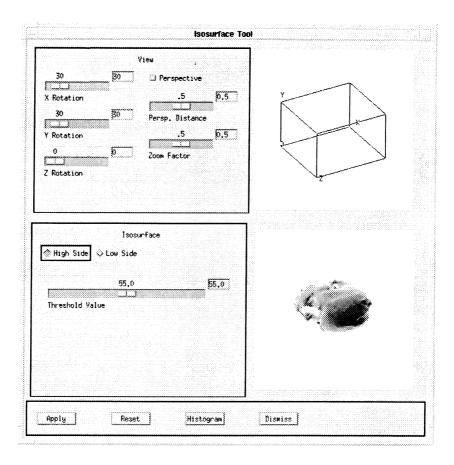


Figure 2-73 WglsoSurfTool creates an interactive window that lets you use the mouse to easily view and modify an iso-surface taken from a three-dimensional block of data. An iso-surface is a pseudo-surface of constant density within a volumetric data set.

Contents of the Window

The IsoSurfTool window has two main parts — the display area and the control area.

IsoSurfTool Display Area

The display area is towards the right of the window; it is divided into an upper and lower region:

- View orientation area The cube establishes a frame of reference for viewing the iso-surface.
- **Iso-surface area** This region contains an image displaying an iso-surface; the iso-surface shows similarities in whatever property the data is measuring.



The orientation of the iso-surface is copied from the orientation of the cube.

IsoSurfTool Control Area

Use the following controls to operate the IsoSurfTool window:

- X, Y, and Z Rotation These controls allow you to rotate the cube (counter-clockwise around the desired axis) a specified number of degrees. The current rotation is shown in the text field to the right of the slider. To modify the rotation, either enter a new value in one of the text fields, or use the left mouse button to drag one of the sliders. If you enter a new value into a text field, press < Return > to apply the new value to the slider and the surface. If you use the sliders, the change is applied immediately as the slider moves.
- **Perspective** If enabled, the cube is drawn with perspective; if disabled, the cube is drawn without perspective.
- Perspective Distance Controls the amount of perspective used when drawing the cube; in other words, how close the eyepoint is to the cube. The closer the eyepoint gets to the cube, the greater the amount of perspective exaggeration that is used to draw the cube.

- **Zoom Factor** Controls the amount of magnification used to draw the cube.
- High side/low side If High Side is enabled, only data values above the threshold value are highlighted on the surface; if Low Side is enabled, only data values below the threshold value are highlighted.

For more information about High Side/Low Side, refer to the description of the *Low* keyword for the SHADE_VOLUME procedure in the *PV*-WAVE Reference.

- Threshold Value Only data values above or below this value are highlighted on the surface, depending on whether High Side or Low Side is enabled.
- **Apply** Redraw the iso-surface with the specified viewing parameters.
- Reset Return to the default viewing parameters and the initial threshold value.
- **Histogram** Draw a histogram of *surface_data* in the isosurface display area. Viewing this histogram can be useful in determining where to place the threshold value.
- **Dismiss** Destroy the IsoSurfTool window and erase it from the screen.



When you click Apply, the graphics you see being redrawn in the iso-surface portion of the display area are being sent to this window via the Z-buffer virtual graphics device. This reduces the time required to redraw the iso-surface by about fourfold. For more information on the Z-buffer graphics device, refer to Z-buffer Output on page A-99 of the PV-WAVE User's Guide.

Event Handling

You can use the IsoSurfTool widget in one of the following two ways:

• From the WAVE> prompt — Enter the procedure name at the WAVE> prompt to display the IsoSurfTool widget. The

IsoSurfTool widget handles its own event loop by calling WwLoop.

Standalone widget in its own window created by another application — The IsoSurfTool widget has its own Main window, but the application (not the IsoSurfTool widget) handles the event loop by calling WwLoop.

The output parameter shell can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgIsoSurfTool is created as a child of parent; otherwise, WgIsoSurfTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgIsoSurfTool window, close it by clicking the Dismiss button.

```
PRO Sample wgisosurftool, parent, tool_shell
head = BYTARR(115, 75, 105)
   Define a variable to hold the data.
IF !Version.platform EQ 'VMS' THEN $
   OPENR, u, GETENV('WAVE_DIR')+$
   '[data]man head.dat', /Get_lun $
ELSE $
OPENR, u, '$WAVE DIR/data/man head.dat', $
   /Get lun
READU, u, head
   Read the PV=WAVE man head.dat demo file that contains
   three-dimensional volumetric data.
CLOSE, u
FREE LUN, u
   Close the file and free the LUN.
```

reduced_head = REBIN(head, 23, 15, 21)

Sample the data so fewer data points are passed in to
WglsoSurfTool. Doing this improves performance significantly,
because WglsoSurfTool has fewer polygons to process.

IF N_ELEMENTS(parent) NE 0 THEN BEGIN
WgIsoSurfTool, reduced_head, parent, \$
 tool shell

Create WglsoSurfTool as a child of the widget known as "parent". The window of the newly created widget is returned via the optional output parameter "tool shell".

ENDIF ELSE BEGIN

WgIsoSurfTool, reduced_head

Create WglsoSurfTool and display it as its own Main window. In other words, the WglsoSurfTool window runs on its own (i.e., in its own event loop).

ENDELSE END

See Also

SHADE_VOLUME, RENDER

For more information about how the UNIX environment variable \$WAVE_GUI or the VMS logical WAVE_GUI affects whether the IsoSurfTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For information about drawing iso-surfaces using voxel data, see Chapter 6, Advanced Rendering Techniques, in the PV-WAVE User's Guide. This chapter includes a number of examples showing iso-surfaces that have been drawn using the RENDER function.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*. For more information about how to write an application program based on PV-WAVE's

Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgMovieTool Procedure

Creates a window that cycles through a sequence of images.

Usage

WgMovieTool, image data [, parent [, shell [, windowid [, rate]]]]

Input Parameters

image_data - A 3D array of images (byte). The dimensions of the array are (m, n, n_frames) , where (m,n) is the size of an individual image, and *n_frames* is the total number of frames in the sequence.

parent — (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgMovieTool runs on its own (i.e., in its own event loop).

rate — (optional) Minimum cycling speed, specified in frames per second; the exact cycling speed varies depending on system load. (This number corresponds to the numeral "1" underneath the speed slider.) By default, under "optimum" conditions, the rate is 30 frames per second (if Pixmap is present and nonzero) or 3 frames per second (if *Pixmap* is not supplied or is equal to zero).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

windowid - (optional) The window ID of the PV-WAVE graphics window. (For information on window IDs, see the description for the WINDOW procedure.)

Input Keywords

Title — A string containing the title that appears in the header of the MovieTool window. Default value is "Movie Tool".

Cmap — The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the MovieTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are either: 1) if *parent* is present and the *Popup* keyword is not specified or has a value of zero, measured from the upper-left corner of a layout (container) widget or 2) if the *Popup* keyword is present and nonzero (regardless of whether *parent* is present and/or nonzero), measured from the upper-left corner of the screen.

Order — The order in which the image is drawn. If present and nonzero, the image is inverted. In other words, the image is drawn from bottom to top instead of from top to bottom.

Pixmap − 1 indicates that pixmaps should be used for the animation; 0 indicates that the data is stored in a PV-WAVE variable. Using pixmaps dramatically improves the speed of the animation, but requires more memory from the X Window System's X server than when data stored in variables is used.

 $Do_tvscl-1$ indicates that TVSCL should be used to scale the image values to the current color table; 0 indicates that TV will be used instead (no scaling).

Maximum — Maximum cycling speed, specified in frames per second; the exact cycling speed varies depending on system load. (This number corresponds to the label to the right of the speed slider.) If **Maximum** is not specified, the maximum rate is 100 frames per second (if **Do_tvscl** is not supplied or is equal to zero) or 10 frames per second (if **Do_tvscl** is present and nonzero).

File — A string containing the name of the file from which the image data is read. When using the File keyword, the Dims keyword must also be supplied. If present, the input variable image_data is ignored.

Dims – A three-element vector specifying the size of the images to be read from the file, and the number of images to read. The elements of the vector are $[m, n, n_frames]$, where (m,n) is the size of an individual image, and *n* frames is the total number of frames in the sequence.

Popup — If present and nonzero, the MovieTool widget is displayed in its own Main window.

Size — A two-element vector specifying the width and height of the display area (long integer). If not specified, the default size of the display area is m-by-n pixels, where m and n are defined by image_data.

View — A two-element vector specifying the width and height of the viewport onto the display area (long integer). If not specified, the default size of the viewport is m-by-n pixels, where m and n are defined by image data.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the color bar widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the movie widget is attached to the bottom of the parent widget.

Left - If a widget ID is specified (for example, Left=wid), then the left side of the movie widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the movie widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the movie widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the movie widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the movie widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the movie widget is attached to the top of the parent widget.

Discussion

WgMovieTool is an interactive window that lets you use the mouse to control the pace and direction of an animated series of images.



Using the WgMovieTool window is similar in many ways to using the WgAnimateTool window, but WgMovieTool is designed so that it can be included inside larger layout widgets, while WgAnimate Tool is intended to be used as a standalone utility widget. Be sure to familiarize yourself with the attachment keywords if you plan to place WgMovieTool inside a larger layout widget.

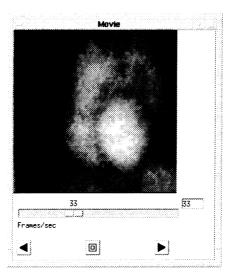


Figure 2-74 WgMovieTool creates an interactive window that lets you use the mouse to control the pace and direction of an animated series of images.

MovieTool provides a graphical user interface (GUI) to a Standard Library procedure, movie.pro. Unlike the blocking behavior that you encounter with movie.pro, you can interact with other windows while MovieTool is open and running.

Contents of the Window

The MovieTool window has two main parts — the display area and the control area.

MovieTool Display Area

The display area is the largest area of the window; it is where the animation sequence takes place.

MovieTool Control Area

Use the following controls to operate the MovieTool window:

Movie speed slider — Decrease or increase the rate of display.

The number shown to the left of the slider is the minimum cycling speed for the animation sequence; this number is controlled with the Rate keyword. The number shown to the right of the slider is the maximum cycling speed for the animation sequence; this number is controlled with the Maximum keyword.

- **Left-pointing arrow** Cycle through the images from last to first.
- **Stop button** Freeze display at the current image.
- **Right-pointing arrow** Cycle through the images from first to last.



Use the window manager menu of the window frame to dismiss the MovieTool window from the screen.

Event Handling

You can use the MovieTool widget in one of the following three ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the MovieTool widget. The MovieTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another **application** — If the *Popup* keyword is present and nonzero, the MovieTool widget has its own Main window. The application (not the MovieTool widget) handles the event loop by calling WwLoop.
- Combined with other widgets in a layout widget Another application combines the MovieTool widget with other widgets inside a layout widget. The application (not the

MovieTool widget) handles the event loop by calling WwLoop.

When the input parameter parent is present and nonzero, the widget operates in either the second or third mode listed, depending on the value of the *Popup* keyword. The output parameter *shell* can be returned only if you also supply the input parameter *parent*.

You can use the output parameter windowid to keep track of the PV-WAVE window ID that is assigned to the MovieTool. You can create multiple instances of the MovieTool; each one will be assigned a different PV-WAVE window ID.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgMovieTool is created as a child of parent; otherwise, WgMovieTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgMovieTool window, close it using the window manager menu.

```
PRO Sample_wgmovietool, parent, tool shell
heart = BYTARR(256, 256, 15)
   Define a variable to hold the data.
IF !Version.platform EO 'VMS' THEN $
   OPENR, u, GETENV('WAVE_DIR')+$
   '[data]heartbeat.dat', /Get lun $
ENDIF ELSE BEGIN
OPENR, u, !Dir+'/data/heartbeat.dat', $
   /Get lun
ENDELSE
READU, u, heart
   Read the PV=WAVE heartbeat.dat demo file that contains
   images showing a beating human heart.
CLOSE, u
FREE LUN, u
   Close the file and free the LUN.
```

IF N_ELEMENTS(parent) NE 0 THEN BEGIN

WgMovieTool, heart, parent, tool_shell, \$
 /Do_tvscl, /Pixmap, /Popup

Create WgMovieTool as a child of the widget known as "parent".

The window of the newly created widget is returned via the optional output parameter "tool shell".

ENDIF ELSE BEGIN

WgMovieTool, heart, /Do_tvscl, /Pixmap Create WgMovieTool and display it as its own Main window. In other words, the WgMovieTool window runs on its own (i.e., in its own event loop).

ENDELSE END

See Also

MOVIE, WgAnimateTool

For more information about how the UNIX environment variable \$WAVE_GUI or the VMS logical WAVE_GUI affects whether the MovieTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about how to transfer image data to PV-WAVE variables, refer to *Input and Output of Image Data* on page 189 of the *PV-WAVE Programmer's Guide*.

For more information about pixmaps, refer to *Using Pixmaps to Improve Application Performance* on page A-85 of the *PV*-WAVE User's Guide.

For more information about color table indices, refer to Experimenting with Different Color Tables on page 310 of the PV-WAVE User's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*. For more information about how to write an application program based on PV-WAVE's

Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgSimageTool Procedure

Creates two windows: 1) a scrolling image window and 2) an optional smaller window that shows a reduced view of the entire image.

Usage

WgSimageTool, image data [, parent [, shell]]

Input Parameters

image_data - A 2D variable containing image data for a single image (byte).

parent — (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgSimageTool runs on its own (i.e., in its own event loop).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the window that is used to display the scrolled image. Default value is "Full Size Image".

Rtitle - A string containing the title that appears in the header of the small window that is used to display the entire image. Default value is "Reduced Image".

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the SimageTool window (long

integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

Order – The order in which the image is drawn. If present and nonzero, the image is inverted. In other words, the image is drawn from bottom to top instead of from top to bottom.

Do_tvscl − 1 indicates that TVSCL should be used to scale the image data before it is displayed; 0 indicates that TV will be used to display the data, unscaled.

Noreduce - If present and nonzero, the reduced size window that is used to display the entire image is not displayed. Only the larger of the two windows is displayed.

Reduced_size — A two-element vector specifying the width and height of the image display area of the reduced size window that displays the entire image (long integer). The reduced size image is computed using CONGRID; if Reduced is not specified, the default size of the reduced image is 256-by-256. This keyword is ignored if *Noreduce* is present and nonzero.

Wsize — A two-element vector specifying the width and height of the image display area of the scrolled image window (long integer). Default display area size is either: 1) 512-by-512, or 2) the size of image data, whichever is less.

Dsize — A two-element vector specifying the width and height of the image to be displayed in the scrolling area (long integer). Default is for *Dsize* to be the same as *image data* in both directions. If Dsize is larger than Wsize, you will be able to use scroll bars to move the image around in the display window. An error occurs if *Dsize* is smaller than the size of *image data* in either direction.



When Dsize is greater than Wsize, and scroll bars are placed on the right and at the bottom of the scrolled image window, then 12 pixels are subtracted in each direction (from the displayed image area) to allow room for the scroll bars.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

WgSimageTool provides a convenient way to view images, either full-size or reduced-size. Scroll bars are provided for viewing large images when they are viewed at full size.

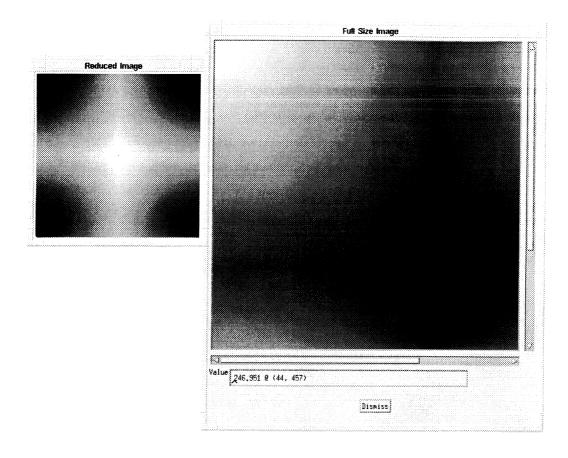


Figure 2-75 By default, WgSimageTool displays two windows. One window displays the image at full-size, and adds scroll bars to the right and bottom for viewing other portions of the image. The other window displays the image at a reduced size (256-by-256 by default) and does not offer the option of scroll bars. Use the Noreduce keyword to suppress the reduced-size window and display only the larger window.

Contents of the Window

The larger of the two SimageTool windows has two main parts the display area and the control area.

SimageTool Display Area

The display area is the largest area of the window; it is where the full-size image is displayed.

SimageTool Control Area

Use the following controls to operate the SimageTool window:

- Value If the pointer is within the boundaries of the image, its location is echoed in this area, along with the value of the image data directly underneath the pointer.
- **Dismiss** Destroy the SimageTool window(s) and erase it (them) from the screen.

Event Handling

You can use the SimageTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the SimageTool widget. The SimageTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another application — The SimageTool widget has its own Main window, but the application (not the SimageTool widget) handles the event loop by calling WwLoop.

The output parameter *shell* can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgSimageTool is created as a child of parent; otherwise, WgSimageTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgSimageTool window, close it by clicking on the Dismiss button.

```
PRO Sample_wgsimagetool, parent, tool shell
x = dist(7000)
   Create a "dummy" variable to use as data for WgSimageTool.
```

IF N ELEMENTS(parent) NE O THEN BEGIN WgSimageTool, x, parent, tool shell, /Do tvscl Create WgSimageTool as a child of the widget known as "parent". The window of the newly created widget is returned via the optional output parameter "tool shell".

ENDIF ELSE BEGIN

WgSimageTool, x, /Do_tvscl

Create WgSimageTool and display it as its own Main window. In other words, the WgSimageTool window runs on its own (i.e., in its own event loop).

ENDELSE END

See Also

TV, TVSCL

For more information about how the UNIX environment variable \$WAVE GUI or the VMS logical WAVE GUI affects whether the Simage Tool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide. For more information

about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgSliceTool Procedure

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.

Usage

WgSliceTool, block data [, parent [, last_slice [, shell]]]

Input Parameters

block data — A 3D variable containing volumetric data. The dimensions of block data define the size of the cube that you see when the SliceTool window is first opened.

parent — (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgSliceTool runs on its own (i.e., in its own event loop).

Output Parameters

last_slice - (optional) The two-dimensional set of values describing the most recent slice that was displayed. The data type of *last slice* is the same as the data type for *block data*.

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the SliceTool window. Default value is "Slicer Tool".

Cmap — The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Cube color — The color in which the rotatable cube is drawn; a positive integer in the range $\{0...255\}$. The number that is provided for cube color is used as an index into the current color table.

Slice color — The color in which the slice is drawn; a positive integer in the range $\{0...255\}$. The number that is provided for slice color is used as an index into the current color table.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the SliceTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

View rot — A three-element vector specifying the initial view rotation, in degrees. If not supplied, the initial rotation of the cube is [30, 30, 0] — in other words, 30 degrees rotation around the X axis, 30 degrees rotation around the Y axis, and 0 degrees rotation around the Z axis.

View zoom — The initial zoom factor; default value is 0.5.

View_persp — The initial perspective distance. If View persp is zero, then the initial setting on the Perspective Distance slider is 0.5, but the Perspective pushbutton is initially deselected. If View_persp is nonzero, then the initial setting on the Perspective Distance slider is the value specified by View persp and the Perspective pushbutton is selected.



Using no perspective to draw the cube is equivalent to having the eyepoint an infinite distance away from the cube, and will produce a cube with sides that all appear to be parallel to one another.

Slice rot — A three-element vector specifying the XYZ slicing plane rotation, in degrees. If not supplied, the initial rotation is [0, 0, 0] — in other words, no rotation.

Slice trans — A three-element vector specifying the initial slicing plane translation, in degrees. If not supplied, the initial translation is [0, 0, 0] — in other words, no translation.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font - Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

WgSliceTool is an interactive window that lets you use the mouse to view and modify the location of a slice that bisects a volume of data; you can position the slice anywhere within the threedimensional input dataset, surface data.

If the zoom factor is large, or the perspective parameter is small, then the cube display in the View Orientation area may be erroneous. To cure the problem, reduce the zoom or increase the perspective (or disable the perspective entirely).

To select a slice, adjust the sliders that control perspective, zoom, and orientation of the slice in relationship to the cube. When you are satisfied with your selections, click the Apply button.



Slices of data are displayed "head-on". In other words, the data slice is not projected into 3D space — the data displayed is the actual 2D slice that you selected.

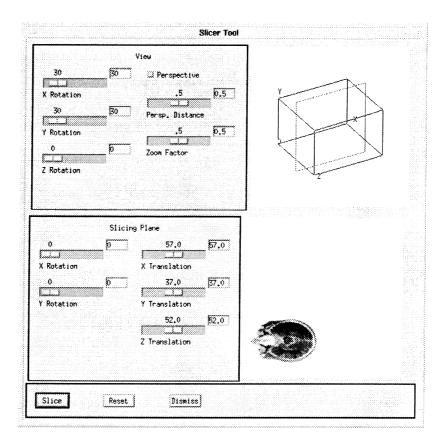


Figure 2-76 WgSliceTool is an interactive window that lets you use the mouse to view and modify the location of a slice that bisects a volume of data; your selections in the lower left part of the window affect the position of the slice.

SliceTool Display Area

The display area is towards the right of the window; it is divided into an upper and lower region:

- View orientation area The cube establishes a frame of reference for positioning the slice of data.
- **Data slice area** This region contains an image displaying the actual slice of data that you have selected for viewing.

Note /

The orientation of the cube provides a frame of reference for selecting and understanding the slice of data that you have chosen to view.

SliceTool Control Area

Use the following controls to operate the SliceTool window:

- X, Y, and Z Rotation These controls allow you to rotate the cube (counter-clockwise around the desired axis) a specified number of degrees. The current rotation is shown in the text field to the right of the slider. To modify the rotation, either enter a new value in one of the text fields, or use the left mouse button to drag one of the sliders. If you enter a new value into a text field, press <Return> to apply the new value to the slider and the surface. If you use the sliders, the change is applied immediately as the slider moves.
- **Perspective** If enabled, the cube is drawn with perspective; if disabled, the cube is drawn without perspective.
- **Perspective Distance** Controls the amount of perspective used when drawing the cube; in other words, how close the eyepoint is to the cube. The closer the eyepoint gets to the cube, the greater the amount of perspective exaggeration that is used to draw the cube.
- **Zoom Factor** Controls the amount of magnification used to draw the cube.
- **Apply** Redraw the cube and the data slice with the specified viewing parameters.
- **Reset** Return to the default viewing parameters and the initial orientation and positioning of the slice.
- **Dismiss** Destroy the SliceTool window and erase it from the screen.

Event Handling

You can use the SliceTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the SliceTool widget. The SliceTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another application - The SliceTool widget has its own Main window, but the application (not the SliceTool widget) handles the event loop by calling WwLoop.

The output parameter *shell* can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgSliceTool is created as a child of parent; otherwise, WgSliceTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgSliceTool window, close it by clicking on the Dismiss button.

```
PRO Sample_wgslicetool, parent, tool_shell
head = BYTARR(115, 75, 105)
   Define a variable to hold the data.
IF !Version.platform EQ 'VMS' THEN $
   OPENR, u, GETENV('WAVE DIR')+$
    '[data]man_head.dat', /Get_lun $
ELSE $
OPENR, u, '$WAVE DIR/data/man head.dat', $
   /Get lun
READU, u, head
   Read the PV=WAVE man head.dat demo file that contains
   three-dimensional volumetric data.
CLOSE, u
```

FREE LUN, u

Close the file and free the LUN.

IF N ELEMENTS(parent) NE O THEN BEGIN

WgSliceTool, head, slice, parent, tool_shell Create WgSliceTool as a child of the widget known as "parent". The window of the newly created widget is returned via the optional output parameter "tool shell".

ENDIF ELSE BEGIN

WgSliceTool, head, slice

Create WqSliceTool and display it as its own Main window. In other words, the WgSliceTool window runs on its own (i.e., in its own event loop).

ENDELSE

END

See Also

VIEWER, SLICE VOL

For more information about how the UNIX environment variable SWAVE GUI or the VMS logical WAVE GUI affects whether the SliceTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For information about other approaches to slicing volumes, see Chapter 6, Advanced Rendering Techniques, in the PV-WAVE User's Guide. This chapter includes an example showing how to render selected slices from a large amount of volume data.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide. For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgStripTool Procedure

Creates a window that displays data in a style that simulates a realtime, moving strip chart.

Usage

```
WgStripTool [, x, y_1, y_2, ..., y_{10}, parent [, shell ]]
WgStripTool [, x [, y_1 [, y_2
      [, y_3 [, y_4 [, y_5 [, y_6 [, y_7 [, y_8 [, y_9 [, y_{10} ]]]]]]]]]]
```

Input Parameters

x – (optional) A vector specifying the X values (horizontal) of each strip chart. If not specified, or set equal to zero, monotonically increasing values, starting with 0 (zero) are automatically provided along the X axis. Must have the same number of elements as the y vectors.

 y_n – (optional) Up to ten vectors containing data in the Y direction (vertical); each vector is displayed in a separate strip chart. The number of y variables specified determines the number of strip charts that are displayed. All y vectors should contain the same number of elements.

parent — (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgStripTool runs on its own (i.e., in its own event loop).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the window. Default value is "Stripchart Tool".

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the WgStripTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

Delay - The minimum interval for updating the strip chart, specified in milliseconds (floating point). The default is to have 0 milliseconds of delay.

Size - A two-element vector specifying the width and height of the display area (long integer); the display area is divided equally between the strip charts for all specified y variables. If not specified, the default size of the display area is 512-by-512 pixels.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font — Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

The WgStripTool window allows you to view data in moving strip charts; you can adjust the display width in the strip charts to create the effect of zooming in and out. You can also adjust the rate and the manner with which the strip charts progress through the data.

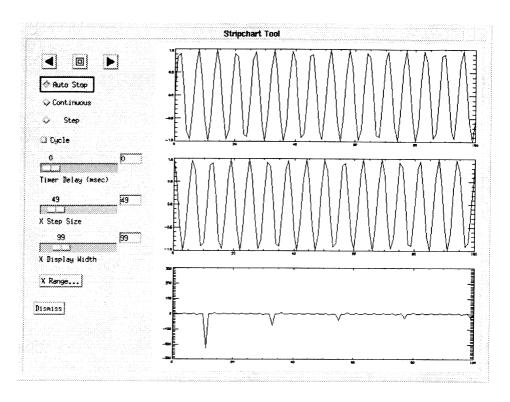


Figure 2-77 The WgStripTool window lets you interactively view data in moving strip charts; up to ten strip charts can be simultaneously displayed by the WgStripTool window. You can adjust the display width in the strip charts to create the effect of zooming in and out. You can also adjust the rate and the manner with which the strip charts progress through the data.

Contents of the Window

The StripTool window has two main parts — the display area and the control area.

StripTool Display Area

The display area is the largest area of the window; it is where the data strip charts are displayed. The number of strip charts you see

in this area is determined by the number of y variables that were included in the call that invoked the WgStripTool window.

StripTool Control Area

Use the following controls to operate the WgStripTool window:

- **Reverse button** Cycle through the data in a reverse direction.
- **Stop button** Freeze strip chart display at current location.
- Forward button Cycle through the data in a forward direction.
- **Mode** Select Auto Stop to have WgStripTool stop whenever it reaches the beginning or the end of the data in the y variable(s). Select Continuous to have WgStripTool move through the data in a non-stop manner, pausing only when you click the Stop button. Select Step to have the strip chart proceed only when you click on one of the arrow buttons.
- Timer Delay Decrease or increase the rate of display. The number shown to the right of the slider is the number of milliseconds delay between contiguous frames of the strip chart.
- X Step Size The amount by which the strip charts are moved forward or back when the strip charts are updated.
- X Display Width The range of data displayed in the horizontal direction. A small value for display width has the effect of "zooming in" to view a narrow range of data; a large value for display width has the effect of "zooming out" to view a broader range of data.
- X Range A more precise way to specify X Display Width. Click this button to display another dialog box containing text fields where you can type precise values for the minimum and maximum values you wish to have displayed along the X axis.
 - This other dialog box is shown in Figure 2-78.
- Dismiss Destroy the WgStripTool window and erase it from the screen.



The selections you make in the control area affect all displayed strip charts. This way, you can be assured that the data in one strip chart is in "sync" with the data in any other strip chart.

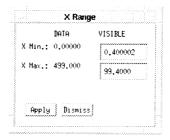


Figure 2-78 Use this dialog box to precisely control the range of the X axis in the WgStripTool window. This dialog box is displayed if you click the X Range button in the WgStripTool control area.

Event Handling

You can use the WgStripTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the WgStripTool widget. The WgStripTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another application The WgStripTool widget has its own Main window, but the application (not the WgStripTool widget) handles the event loop by calling WwLoop.

The output parameter *shell* can be returned only if you also supply the input parameter *parent*.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgStripTool is created as a child of parent; otherwise, WgStripTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgStripTool window, close it by clicking on the Dismiss button.

```
PRO Sample wgstriptool, parent, tool_shell
x = indgen(500)
y1 = sin(x)
y2 = cos(x)
y3 = tan(x)
```

Create an independent variable and three dependent variables describing sinusoidal curves to use as data for WgStripTool.

```
IF N ELEMENTS(parent) NE 0 THEN BEGIN
WgStripTool, x, y1, y2, y3, $
    0, 0, 0, 0, 0, 0, 0, parent, tool shell
        Create WgStripTool as a child of the widget known as
        "parent". The window of the newly created widget is
        returned via the optional output parameter "tool_shell".
```

ENDIF ELSE BEGIN

```
WgStripTool, x, y1, y2, y3
```

Create WgStripTool and display it as its own Main window. In other words, the WgStripTool window runs on its own (i.e., in its own event loop). Notice how the zeroes are not needed as placeholders, since the optional parameters "parent" and "tool_shell" are not being used in this procedure call.

ENDELSE END

See Also

For more information about how the UNIX environment variable SWAVE GUI or the VMS logical WAVE GUI affects whether the WgStripTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide. For more information about how to write an application program based on PV-WAVE's

Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgSurfaceTool Procedure

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.

Usage

WgSurfaceTool, surface data [, parent [, shell]]

Input Parameters

surface_data - A 2D variable containing surface data.

parent — (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgSurfaceTool runs on its own (i.e., in its own event loop).

Output Parameters

shell – (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the Surface Tool window. Default value is "Surface Tool".

Cmap – The index of the color table to load when the widget is created; a positive integer in the range $\{0...15\}$.

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the SurfaceTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

Nogrid — If present and nonzero, no grid is drawn on the surface.

Lines – If present and nonzero, lines are drawn instead of a grid.

Upper — If present and nonzero, only the upper portion of the grid is drawn. Upper and Lower are mutually exclusive keywords.

Lower — If present and nonzero, only the lower portion of the grid is drawn. Lower and Upper are mutually exclusive keywords.

Skirt – If present and nonzero, a skirt is drawn connecting the surface to the X and Y axes. Refer to Figure 2-80 for an example of a surface with and without a skirt.

Gouraud — If present and nonzero, the surface is drawn using Gouraud shading. Gouraud and Elevation are mutually exclusive.

Elevation — If present and nonzero, the surface is drawn using simple elevation shading. Elevation and Gouraud are mutually exclusive.

Xrot — The initial counter-clockwise rotation of the surface around the X axis, measured in degrees.

Zrot – The initial counter-clockwise rotation of the surface around the Z axis, measured in degrees.

Auto redraw — If present and nonzero, the surface is automatically redrawn any time one of the surface parameters is adjusted.

Auto congrid — If present and nonzero, the Z variable is resized to 50-by-50 (with CONGRID) prior to drawing the surface. This enables PV-WAVE to draw the surface much faster, although there is a chance that some surface detail is lost.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor – Specifies the base color.

Font - Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

WgSurfaceTool is an interactive window that lets you use the mouse to control the orientation and appearance of a displayed surface.

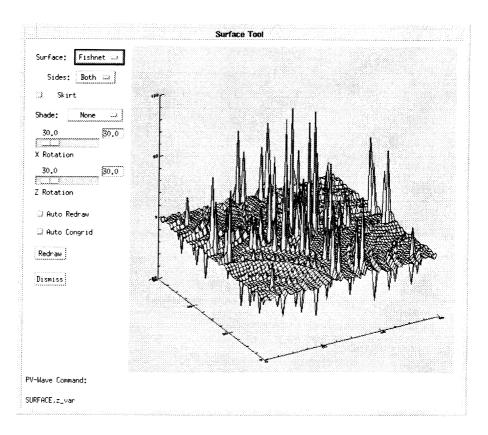


Figure 2-79 WgSurfaceTool creates an interactive window that lets you use the mouse to control the orientation and appearance of a displayed surface.

Contents of the Window

The SurfaceTool window has three main parts — the display area, the control area, and the message area.

SurfaceTool Display Area

The display area is the largest area of the window; it is where the surface is displayed.

SurfaceTool Control Area

Use the following controls to operate the SurfaceTool window:

- **Surface (menu)** From the menu, choose the method by which the surface is drawn. The choices are: Fishnet (mesh), Lines (lines in one direction only), and None (no lines appear superimposed on a shaded surface).
- **Sides (menu)** From the menu, choose the representation of the top and bottom sides of the surface. The choices are: Both (lines on both top and bottom), Upper (lines on upper surface only), and Lower (lines on lower surface only).
- **Skirt** Controls whether a skirt is added to the surface. A skirt helps establish a frame of reference between the surface and the X, Y, and Z axes. Refer to Figure 2-80 for an example of a surface with and without a skirt.
- **Shade (menu)** From the menu, select the algorithm by which the surface is shaded. The choices are: None (no shading), Gouraud, and Elevation.
- **X and Z Rotation** These controls allow you to rotate the surface (counter-clockwise around the X or Z axis) a specified number of degrees. The current rotation is shown in the text field to the right of the slider. To modify the rotation, either enter a new value in one of the text fields, or use either slider. If you enter a new value into a text field, press <Return> to apply the new value to the slider and the surface. If you use the sliders, the change is applied immediately as the slider moves.

- Auto Redraw Controls whether the contents of the display area are redrawn every time a modification is made to one of the controls in the SurfaceTool control area.
- Auto Congrid The surface is resized to 50-by-50 (with CONGRID) prior to drawing the surface, for faster display of large datasets. The default is for Auto Congrid to be enabled. Auto Congrid does not affect the actual data; it only affects the display of the data.
- **Redraw** Causes the contents of the SurfaceTool display area to be redrawn.
- **Dismiss** Destroy the SurfaceTool window and erase it from the screen.

SurfaceTool Message Area

The message area displays the PV-WAVE command that is being used to display the surface.

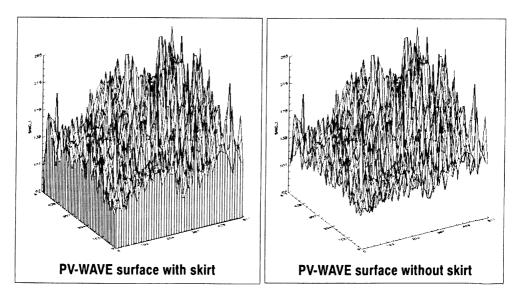


Figure 2-80 A surface with and without a skirt.

Event Handling

You can use the SurfaceTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the SurfaceTool widget. The SurfaceTool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another application — The SurfaceTool widget has its own Main window, but the application (not the SurfaceTool widget) handles the event loop by calling WwLoop.

The output parameter shell can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgSurfaceTool is created as a child of parent; otherwise, WgSurfaceTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgSurfaceTool window, close it by clicking on the Dismiss button.

```
PRO Sample wgsurfacetool, parent, tool_shell
x = dist(75)
```

Create a "dummy" variable to view as view as a surface using WgSurfaceTool.

```
IF N ELEMENTS(parent) NE 0 THEN BEGIN
WqSurfaceTool, x, parent, tool shell
```

Create WgSurfaceTool as a child of the widget known as "parent". The window of the newly created widget is returned via the optional output parameter "tool_shell".

ENDIF ELSE BEGIN

WgSurfaceTool, x

Create WgSurfaceTool and display it as its own Main window. In other words, the WgSurfaceTool window runs on its own (i.e., in its own event loop).

ENDELSE END

See Also

SURFACE, SHADE SURF

For more information about how the UNIX environment variable \$WAVE_GUI or the VMS logical WAVE GUI affects whether the SurfaceTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide. For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WgTextTool Procedure

Creates a scrolling window for viewing text from a file or character string.

Usage

WgTextTool [, parent [, shell]]

Input Parameters

parent – (optional) The widget or shell ID of the parent widget (long). If parent is not specified, WgTextTool runs on its own (i.e., in its own event loop).



Either the File or the Text keyword must be supplied to identify the text that WgTextTool will display.

Output Parameters

shell - (optional) The ID of the newly created widget. If the procedure fails, zero (0) is returned.

Input Keywords

Title - A string containing the title that appears in the header of the TextTool window. If not specified, either the name of the file is used for the title, or when TextTool is displaying a string, the value of the title defaults to "Scrolling Window".

Position — A two-element vector specifying the X and Y coordinates of the upper-left corner of the TextTool window (long integer). The elements of the vector are [x, y], where x (horizontal) and y (vertical) are specified in pixels. These coordinates are measured from the upper-left corner of the screen.

File — The name of the file to display. The File and Text keywords are mutually exclusive, and one of them must be used.

Text — A string containing the text that will be displayed. The **Text** and **File** keywords are mutually exclusive, and one of them must be used.

Rows — The number of text rows to display when the window is first created.

Cols — The number of text columns to display when the window is first created.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor – Specifies the base color.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Discussion

WgTextTool is a window that lets you view text from a string or from a file.

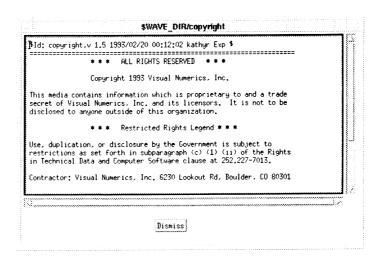


Figure 2-81 WgTextTool displays either: 1) text from a file, or 2) text from an input string. In this illustration, WgTextTool is displaying text from the copyright file that is distributed with every copy of PV=WAVE.



The text displayed in the window is read only; it cannot be edited.

Interacting with the Window

The standard text editing functions are available. In Motif, this means the left mouse button and the middle mouse button can be used to copy text. In OPEN LOOK, this means that the text edit popup menu is available. However, in the text edit popup menu, only the Copy function will be sensitized, because the text is readonly.

Use the Dismiss button (or the TextTool window's window manager menu) to destroy the window when you are finished viewing the text.

Event Handling

You can use the TextTool widget in one of the following two ways:

- From the WAVE> prompt Enter the procedure name at the WAVE> prompt to display the TextTool widget. The Text-Tool widget handles its own event loop by calling WwLoop.
- Standalone widget in its own window created by another application - The TextTool widget has its own Main window, but the application (not the TextTool widget) handles the event loop by calling WwLoop.

The output parameter shell can be returned only if you also supply the input parameter parent.

Example

Enter the commands shown below into a file, and compile the procedure with the .RUN command. If the variable parent is defined, WgTextTool is created as a child of parent; otherwise, WgTextTool runs on its own (i.e., in its own event loop).

When you are finished interacting with the WgTextTool window, close it by clicking the Dismiss button.

```
PRO Sample_wgtexttool, parent, tool_shell
IF !Version.platform EQ 'VMS' THEN BEGIN
filename = GETENV('WAVE DIR')+'copyright'
   Specify the pathname and filename using VMS notation.
ENDIF ELSE BEGIN
filename = '$WAVE DIR/copyright'
   Specify the pathname and filename using UNIX notation.
ENDELSE
```

IF N_ELEMENTS(parent) NE 0 THEN BEGIN WgTextTool, File=filename, parent, tool_shell Create WgTextTool as a child of the widget known as "parent". The window of the newly created widget is returned via the optional output parameter "tool shell".

ENDIF ELSE BEGIN

WgTextTool, File=filename

Create WgTextTool and display it as its own Main window. In other words, the WgTextTool window runs on its own (i.e., in its own event loop).

ENDELSE END

See Also

WwText

For more information about how the UNIX environment variable \$WAVE GUI or the VMS logical WAVE GUI affects whether the TextTool window is implemented using the Motif or the OLIT look-and-feel, refer to Specifying the Desired Toolkit on page 411 of the PV-WAVE Programmer's Guide.

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide. For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WHERE Function

Returns a longword vector containing the one-dimensional subscripts of the nonzero elements of the input array.

Usage

result = WHERE(array_expr [, count])

Input Parameters

array_expr — The array to be searched. May be of any data type except string. Both the real and imaginary parts of complex numbers must be zero for the number to be considered zero.

Output Parameters

count — If present, *count* is converted into a longword integer containing the number of nonzero elements found in *array_expr*. Must be a named variable.

Returned Value

result — A longword vector containing the one-dimensional subscripts of the nonzero elements of *array_expr*. The length is equal to the number of nonzero elements in *array_expr*.

Keywords

None.

Discussion

Frequently the result of WHERE is used as a vector subscript to select elements of an array using given criteria.

As a side effect, the system variable !Err is set to the number of nonzero elements. This is for compatibility with previous versions of PV-WAVE. Therefore, it is recommended that the Count keyword be used in all new programs, rather than !Err.

Example 1

The WHERE function can be used to select a range of values in an array. For example:

```
index = WHERE ((array GT 50) AND $
   (array LT 100))
        Get the subscripts of those elements greater than 50 and
        less than 100.
```

```
result = array(index)
   Put the selected values into result.
```

Example 2

If all the elements of array expr are zero, WHERE returns a scalar integer with a value of -1. If you attempt to use this result as an index into another array, you will get an error message about the subscripts being out of bounds. In some situations, code similar to the following can be used to avoid errors:

```
index = WHERE(array, count)
   Use count to get the number of nonzero elements.
IF (count GT 0) THEN result = array(index)
   Only subscript the array if it's safe to do so.
```

See Also

```
!Err, QUERY TABLE, SORT
```

For more information, see Chapter 5, Using Subscripts and Matrices, in the PV-WAVE Programmer's Guide.

WINDOW Procedure

Creates a window for the display of graphics or text.

Usage

WINDOW [, window_index]

Input Parameters

window_index — An integer between 0 and 31 specifying the window index for the newly created window.

- If window_index is omitted, 0 is used.
- If the value of window_index specifies an existing window, the existing window is deleted and a new window is created.

Input Keywords

Colors — The maximum number of color table indices to be used. This keyword has effect only if it is supplied when the first window is created. Otherwise, PV-WAVE uses all of the available color indices.

PV-WAVE maintains one color table for all of its windows. To use monochrome windows on a color display, use Colors=2 when creating the first window.

If the X Window System is being used, a negative value for *Colors* specifies that all but the given number of colors from the shared color table should be allocated.

Free — If nonzero, creates a window using the largest unused window index. This keyword can be used instead of specifying the window_index parameter. The default position of the new window is opposite the current window.

Noretain — An obsolete keyword (see the *Retain* keyword below).

Pixmap — If the current graphics device is the X Window System, this keyword specifies that the window being created is actually an invisible portion of the display memory called a pixmap.

Retain – Specifies how backing store for the window should be handled. Possible values for this keyword are listed below:

- 0 No backing store (same as *Noretain* keyword).
- 1 The server or window system is requested to make the window retained.
- 2 PV-WAVE should provide a backing pixmap and handle the backing store directly (X Window System only).

Set Xpix_ID - (X Window System only) Uses the X pixmap associated with the X Pixmap ID assigned to this keyword for the PV-WAVE pixmap. This keyword forces the *Pixmap* keyword to be set. For example:

```
WINDOW, Set_Xpix_ID=1234567L
```

Set Xwin ID – (X Window System only) Uses the X Window associated with the X Window ID assigned to this keyword for the PV-WAVE window. For example:

```
WINDOW, Set Xwin ID=1234567L
```

Title - A scalar string specifying the window's label. If not specified, the window is given a label of the form "WAVE n", where n is the index number of the window.

For example, to create a window with a label of PV-WAVE CL Graphics:

```
WINDOW, Title='PV-WAVE CL Graphics'
```

Xpos, *Ypos* — The X and Y positions of the lower-left corner of window, specified in device coordinates.

If no position is specified, the position of the window is determined from the value of window index, using the following rules of thumb:

Window 0 is placed in the upper right-hand corner.

- Even numbered windows are placed on the top half of the screen and odd numbered windows are placed on the bottom half.
- Windows 0, 1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28, and 29 are placed on the right side of the screen and windows 2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23,26, 27, 30, 31 are placed on the left.

XSize — The width of the window, in pixels. The default is 640.

YSize — The height of the window, in pixels. The default is 512.

Output Keywords

Get_Xpix_ID — (X Window System only) Returns the X pixmap ID for the pixmap just created. You must use the Pixmap keyword to set this ID. For example:

WINDOW, Get_Xpix_ID=New_Xpix_ID, /Pixmap

Get_Xwin_ID — (X Window System only) Returns the X Window ID for the window just created. For example:

Discussion

You only need to use WINDOW if you want to display more than one PV-WAVE window simultaneously, because a window is created automatically the first time any display procedure attempts to access the window system.

The newly created window becomes the current PV-WAVE window, and the system variable !D.Window is set to the window index associated with it. (See the WSET procedure for a discussion of the current PV-WAVE window.)

The behavior of WINDOW varies slightly depending on the window system in effect.

See Also

!D.Window, WDELETE, WSET, WSHOW, ERASE

For additional information on PV-WAVE graphics devices, see Appendix A, Output Devices and Window Systems, in the PV-WAVE User's Guide.

WMENU Function

Displays a menu inside the current window whose choices are given by the elements of a string array and which returns the index of the user's response.

Usage

result = WMENU(strings)

Input Parameters

strings — Must be a string array, with each element containing the text of one menu choice. Both the maximum number of elements and the maximum element length are constrained by how large a display area will be used.

Returned Value

result - Returns -1 if the user did not select a menu item. Otherwise, the returned value ranges from 0 to the number of elements in strings minus one.

Input Keywords

Initial Selection — The index of the initial selection.

If this keyword is specified and within the range of strings indices, the initial menu display is made with the designated item selected.

 If this keyword is not specified, the menu is initially displayed with the mouse cursor at the immediate left of the first (top) selection.

Title — The index of the *strings* element that is the title, normally 0. The title element is not selectable and is displayed reversed and centered. If this keyword is omitted, all items are selectable.

Xpos - (X Window System only) The position on the X axis of the display device where the menu is to be placed.

Ypos – (X Window System only) The position on the Y axis of the display device where the menu is to be placed.

Discussion

WMENU can be used only with X Window System displays.

To use, select a menu item with the mouse and click the left mouse button.

Example

The following statement displays a menu containing the selections Yes and No and entitled Do you wish to continue?:

```
i = WMENU(['Do you wish to continue?', 'Yes',$
  'No'], Title=0, Init=1)
```

The menu is displayed with Yes initially selected. The result is as follows:

- 1 if the user clicks on Yes.
- 2 if the user clicks on No.
- -1 if the user clicks the left mouse button outside the menu.

See Also

TVMENU

WRITEU Procedure

Writes binary (unformatted) data from an expression into a file.

Usage

```
WRITEU, unit, expr_1, \dots, expr_n
```

Input Parameters

unit – The file unit to which the output will be sent.

expr_i - Expressions to be output. For nonstring variables, the number of bytes contained in expr is output. For string variables, the number of bytes contained in the existing string is output.

Keywords

None.

Discussion

WRITEU performs a direct transfer, with no processing of any kind being done to the data.

Example

In this example, WRITEU is used to write some data to a file. The READU procedure could then be used to read the data from the file.

```
d = BYTSCL(REFORM(FIX(100 * RANDOM(40000)), $
   200, 200))
```

Create some data. Argument d contains a 200-by-200 byte

```
OPENW, unit, 'wuex.dat', /Get_Lun
   Open the file wuex.dat for writing.
```

```
WRITEU, unit, d
```

Write the data in d to the wuex.dat file.

FREE LUN, unit

Close the file and free the file unit number.

See Also

READU, OPENR

For more information and examples, see *READU* and *WRITEU* on page 193 of the *PV*-*WAVE Programmer's Guide*.

WSET Procedure

Used to select the current window to be used by the graphics and imaging routines.

Usage

WSET [, window index]

Input Parameters

window_index - The window index of the new current window.

Keywords

None.

Discussion

WSET can be used only on displays with window systems.

Most PV-WAVE graphics routines do not explicitly require the PV-WAVE window to be specified. Instead, they use the window known as the current window. The window index number of the current window is given by the read-only system variable !D.Window.

See Also

WDELETE, WINDOW, WSHOW, !D.Window

WSHOW Procedure

Exposes or hides the designated window. It does not automatically make the designated window the active window.

Usage

WSHOW [, window_index [, show]]

Input Parameters

window index – The window index of the window to be hidden or exposed. If not specified, the current window is used.

show - A flag indicating whether a window is hidden or exposed:

- 0 Hides the window.
- 1 Exposes the window.

Input Keywords

Iconic – If present and nonzero, turns the window into an icon.

Discussion

The definition of "hidden" is machine-dependent. On Sun Workstations, for example, the window disappears, although it can still be written to if it is the active window.

On machines supporting the X Window System server, the window is simply pushed to the back of the window display list so that it appears to be located behind other windows on the display.

See Also

WDELETE, WINDOW, WSET, !D.Window

WtAddCallback Function

Registers a PV-WAVE callback routine for a given widget.

Usage

status = WtAddCallback(widget, reason, callback [, client data])

Input Parameters

widget – The widget ID of the widget to add the callback to (long).

reason — A string containing the callback reason. This parameter is GUI-dependent. See the Discussion section below for more information.

callback - A string containing the name of the PV-WAVE callback routine.

client_data - A PV-WAVE variable. The value of this variable is passed to the callback routine.

Returned Value

status — One (1) indicates success; zero (0) indicates failure.

Keywords

Noparams – If present and nonzero, the callback is called with two parameters: wid and data. All other parameters, as discussed in Appendix C, Motif and OLIT Callback Parameters, in the PV-WAVE Programmer's Guide.

Discussion

Callback reasons are listed throughout the OSF/Motif Programmer's Reference and the OLIT Programmer's Reference. To use a Motif or OLIT callback reason in PV-WAVE, remove the XmN or XtN prefix. For example:

OLIT Motif PV-WAVE Widget Set

XmNdestroyCallback XtNdestroyCallback destroyCallback

The application can optionally use the *client_data* parameter to specify some application-defined data to be passed to the callback procedure when the callback is invoked. If *client data* is a local variable (defined only in the current procedure), a copy of that variable is created and passed (passed by value). If the client_data is a global variable (defined in a Common Block), it is passed by reference.

Example

This example creates a Motif button labeled Done. When you select the button, the widget is destroyed. To run the example, enter the callback and the example procedures in a file and run them with .RUN.

Callback Procedure Example: Motif

This is the callback routine. Note that the callback routine for the pushbutton widget class requires six parameters. The required callback parameters for Motif widget classes are discussed in Motif Widget Classes on page B-1, in the PV-WAVE Programmer's Guide.

```
PRO CancelHelp, wid, data, npar, reason,$
   event, count
   COMMON block, top
   status=WtClose(top)
END
```

Example Procedure

```
PRO example

COMMON block, top

@wtxmclasses.pro

top=WtInit('wt_ex1', 'Examples')

widget=WtCreate('Done', $

xmPushButtonWidgetClass, top)

status=WtAddCallback(widget, $

'activateCallback', 'CancelHelp')

status=WtSet(top, /Realize)

WtLoop

END
```



The nonstandard callback reason XtNdefaultAction has been added to the OLIT widget scrollingListWidgetClass. This reason was added to support the default action of a double-click to select an item.

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WtAddHandler Function

Registers the X event handler function for a given widget.

Usage

status = WtAddHandler(widget, eventmask, handler [, client data])

Input Parameters

widget — The ID of the widget to add the event handler to (long).

eventmask — The value of the X Event mask (long). See the wtxlib.pro file in the PV-WAVE Standard Library and the Xlib Reference Manual (O'Reilly & Associates, Inc., 1989) for more information on this parameter.

handler - A string containing the PV-WAVE X event handler procedure name.

client data — A PV-WAVE variable. The value of this variable is passed to the callback routine.

Returned Value

status — One (1) indicates success; zero (0) indicates failure.

Keywords

Nonmaskable — If present and nonzero, nonmaskable events are intercepted by the event handler. Such events include GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify.

Discussion

An event handler is a PV-WAVE procedure that is executed when a specific type of event occurs within a widget. Some, all, or no X events can be handled using one or more event handlers.

For information on the requirements for writing event handler procedures, see Adding Event Handlers on page 487 of the PV-WAVE Programmer's Guide.

Example

The following code fragments demonstrate the use of WtAddHandler.

```
pane=WtCreate('menu', PopupMenuWidget, $
   parent)
status = WtAddHandler(pane, ButtonPressMask, $
   'PostMenu', parent)
PRO PostMenu, wid, parent, nparams, mask, event
   @wtxlib
   status=WtPointer("GetLocation", wid, state)
   if (Button3Mask AND state(6)) ne 0 then $
   status=WtSet(pane,POPUP=event)
END
```

X Event Handler Procedure Example

```
PRO handler, widget, data, nparams, mask, $
   event
   . . .
END
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WtClose Function

Closes the current Xt (OLIT or Motif) session, and destroys all children of the top-level widget created in WtInit. This routine can also be used to destroy additional widget trees.

Usage

status = WtClose(widget)

Parameters

widget — The widget ID of the top-level shell (long).

Returned Value

status — One (1) indicates success; zero (0) indicates failure.

Discussion

This function is usually called in a PV-WAVE callback routine to destroy a popup shell (dialog, etc.).

Example

This example creates a Motif button labeled Done. When you select the button, the widget is destroyed. To run the example, enter the callback and the example procedures in a file and run them with .RUN.

Callback Procedure Example: Motif

This is the callback routine. Note that the callback routine for the pushbutton widget class requires six parameters. The required callback parameters for Motif widget classes are discussed in Appendix B, Motif and OLIT Widget Classes, in the PV-WAVE Programmer's Guide.

```
PRO CancelHelp, wid, data, npar, reason,$
    event, count
    COMMON block, top
    status=WtClose(top)
END
```

Example Procedure

```
PRO example

common block, top

@wtxmclasses.pro

top=WtInit('wt_ex2', 'Examples')

widget=WtCreate('Done', $

xmPushButtonWidgetClass, top)

status=WtAddCallback(widget, $

'activateCallback', 'CancelHelp')

status=WtSet(top, /Realize)

WtLoop

END
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WtCreate Function

Creates a widget or shell instance specified by widget class.

Usage

widget = WtCreate(name, class, parent [, argv])

Parameters

name — A string containing the name of the widget or shell to be created.

class — A constant value specifying the widget or shell class (long). See Appendix B, Motif and OLIT Widget Classes, in the PV-WAVE Programmer's Guide for a list of Motif and OLIT widget classes.

parent — The widget or shell ID of the parent (long).

argy — A structure that contains the arguments or resources for the widget or shell.

Returned Value

widget - A newly created widget. If the function fails, zero (0) is returned.

Keywords

Multi — Used with the OLIT scrollingList widget class only. Allows selection of multiple items.



For OLIT flat widgets use argv to specify the attributes of subitems. Use tag names of the argv structure as the resource names.

Example

```
items=REPLICATE({FLATNON,label:'', $
    mnemonic:''},3)
items(0)={FLATNON,'Bold','B'}
items(1)={FLATNON,'Italic','I'}
items(2)={FLATNON,'Underline','U'}
fargs = {,items:items}
widget=WtCreate('fnon', $
    flatNonexclusivesWidgetClass, parent,$
    fargs)
```



All widgets are managed when created. To unmanage them after creation use WtSet(wid,/Unmanage).

See Also

WtSet

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WtCursor

Sets or changes the cursor.

Usage

status = WtCursor(function, widget [, index])

Parameters

function:

'Default' — The default is the system cursor.

'System' — Sets the default system cursor.

'Wait' - Sets the wait cursor.

'Set' - Sets the specified cursor. If 'Set' is specified, the index parameter follows:

index — The cursor index (e.g., XC_X_cursor). See Appendix D, Widget Toolbox Cursors, in the PV-WAVE Programmer's Guide for a list of cursors.

widget - The ID of the widget for which the cursor is being set.

Returned Value

status — One (1) indicates success; zero (0) indicates failure.

Discussion

This routine changes the current cursor for a given widget to a new cursor defined by *index*. The following cursors are available:

- All XC_* cursor types are listed in Appendix D, Widget Toolbox Cursors, in the PV-WAVE Programmer's Guide. For additional information on these cursors, see Appendix I of the Xlib Reference Manual, Volume 2, (O'Reilly & Associates, Inc., 1989).
- A set of custom cursors designed by Visual Numerics listed in Appendix D, Widget Toolbox Cursors, in the PV-WAVE Programmer's Guide.

Example

This example demonstrates a callback called to display the heartbeat.dat file with the WgMovieTool procedure. Because it takes a while to read the data file into PV-WAVE, the wait cursor is set before the file is read to notify the user that the file is being read:

```
PRO MovieCB, wid, index

@wtcursor

top = WwGetValue(wid, /Userdata)

CASE index OF
   1: BEGIN
   status = WtCursor('WAIT', top)
   heart = BYTARR(256, 256, 15)

IF !Version.platform EQ 'vms' THEN $
     OPENR, u, getenv('WAVE_DIR')+ $
        '[data]heartbeat.dat', /Get_Lun $
     ELSE $
     OPENR, u, $
        '$WAVE_DIR/data/heartbeat.dat',$
        /Get_Lun
        READU, u, heart
```

```
CLOSE, u
WgMovieTool, heart, top, movie, $
widx, 1, /Popup, /Do_tvscl
status = WtCursor('DEFAULT', top)
END

2: BEGIN
status = WwSetValue(top, /Close)
END
ENDCASE
END
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WtGet Function

Retrieves widget resources.

Usage

value = WtGet(widget [, resource])

Parameters

widget — The widget ID.

resource — A string containing the name of the requested resource. This parameter is GUI-dependent. See the *Discussion* section below for more information.

Returned Value

value — A PV-WAVE variable in which the value of the resource is returned. The data type of value depends on the requested resource.

Keywords

Child=child — (Motif only) Returns the ID of the child widget in a composite widget, such as a Command or FileSelection widget:

Legal values for a Command widget:

- XmDIALOG_COMMAND TEXT
- XmDIALOG_PROMPT_LABEL
- XmDIALOG HISTORY LIST

Legal values for a FileSelection widget:

- XmDIALOG APPLY BUTTON
- XmDIALOG_CANCEL_BUTTON
- XmDIALOG DEFAULT BUTTON
- XmDIALOG_DIR_LIST

- XmDIALOG_DIR_LIST_LABEL
- XmDIALOG FILTER LABEL
- XmDIALOG FILTER_TEXT
- XmDIALOG HELP BUTTON
- XmDIALOG_LIST
- XmDIALOG_LIST_LABEL
- XmDIALOG_OK_BUTTON
- XmDIALOG_SELECTION_LABEL
- XmDIALOG_SEPARATOR
- XmDIALOG_TEXT
- XmDIALOG_WORK_AREA

For a MessageBox widget:

- XmDIALOG_CANCEL_BUTTON
- XmDIALOG DEFAULT BUTTON
- XmDIALOG_HELP_BUTTON
- XmDIALOG MESSAGE_LABEL
- XmDIALOG_OK_BUTTON
- XmDIALOG_SEPARATOR
- XmDIALOG_SYMBOL_LABEL

For a SelectionBox widget:

- XmDIALOG_APPLY_BUTTON
- XmDIALOG_CANCEL_BUTTON
- XmDIALOG_DEFAULT_BUTTON
- XmDIALOG_HELP_BUTTON
- XmDIALOG_LIST
- XmDIALOG_LIST_LABEL
- XmDIALOG OK BUTTON
- XmDIALOG SELECTION LABEL

- XmDIALOG SEPARATOR
- XmDIALOG_TEXT
- XmDIALOG WORK AREA

Child — Returns a long integer array of child widget IDs.

Class — Returns the widget class for the given widget.

Count=count — Specifies the number of items for resources containing an array of strings, such as a list or command.

Managed — Returns 1 if the given widget is managed; returns 0 if the widget is unmanaged.

Name – Returns the name of the given widget.

Name=name — Returns the ID of the named widget and its parent.

Ncols — Specifies the number of columns of the two-dimensional resource to be retrieved. A two-dimensional resource is a resource whose value is a two-dimensional array of strings. Currently, two-dimensional resources are used in the table widget.

Nrows — Specifies the number of rows of the two-dimensional resource to be retrieved.

Parent — Returns the widget ID of the parent of the given widget.

Realized — Returns 1 if the given widget is realized (displayed); returns 0 if it is unrealized.

Sensitive — Returns 1 if the given widget is sensitive; returns 0 if it is not sensitive.

Shell — Returns 1 if the given widget is a shell (top-level widget); returns 0 if it is not a shell.

Userdata — Returns the user data (any variable or structure) for the given widget formally stored by the WtSet function.

Value — Returns the value for a scale, scroll bar, or toggle button.

Widget — Returns 1 if the given widget is a widget (not a shell); returns 0 if it is not a widget.

Window - Returns the window ID of the given widget.

Window=window - Returns the widget ID of the given window.

Discussion

See the OSF/Motif Programmer's Reference or the OLIT Programmer's Reference for a list of resource names. The resource name for a Widget Toolbox widget is derived from the Motif or OLIT widget set resource name. If you have a Motif toolkit, remove the XmN prefix from the Motif resource name. If you have an OLIT toolkit, remove the XtN prefix from the OLIT resource name. For example:

Motif	OLIT	PV-WAVE Widget Set
XmNwidth	XtNwidth	width
XmNx	XtNx	x

The data type of a resource's value depends on the type of the resource.

Example

```
window = WtGet(wid, /Window)
    Returns the window ID of the specified widget (wid).
```

```
wid = WtGet(wid, Window=win(2))
```

Returns the widget ID of window win(2) for the specified widget (wid).

```
widget = WtGet(wid, /Widget)
```

Returns 1 if the given widget is not a shell, or 0 if it is a shell.

```
name = WtGet(wid, /Name)
```

Returns the name of the specified widget (wid).

child = WtGet(wid, Child=XmDIALOG DIR LIST) Returns the ID of the XMDIALOG DIR LIST component the FileSelection widget (wid).

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WtInit Function

Initializes the Widget Toolbox and the Xt toolkit, opens the display, and creates the first top-level shell.

Usage

topshell = WtInit(app_name, appclass_name [, Xserverargs ...])

Parameters

app_name — A string containing the name of the application as used in the resource file.

appclass_name — A string containing the application class name (name of the resource file).

Xserverargs — A string containing X server arguments (font, display, synchronize, etc.). See the OSF/Motif Programmer's Reference or the OLIT Programmer's Reference for more information.

Returned Value

topshell – Returns the widget ID of the top application shell.

Discussion

Call this routine before the first use of any Widget Toolbox routines, or to reinitialize Widget Toolbox after closing the top-level shell(s).

Example

This example creates a Motif button labeled Done. When you select the button, the widget is destroyed. To run the example, enter the callback and the example procedures in a file and run them with .RUN.

Callback Procedure Example: Motif

```
PRO CancelHelp, wid, data, npar, reason,$
   event, count
   COMMON block, top
   status=WtClose(top)
END
```

Example Procedure

```
PRO example
   COMMON block, top
   @wtxmclasses.pro
   top=WtInit('wt ex3', 'Examples')
   widget=WtCreate('Done', $
      xmPushButtonWidgetClass, top)
   status=WtAddCallback(widget, $
      'activateCallback', 'CancelHelp')
   status=WtSet(top, /Realize)
   WtLoop
END
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WtInput Function

Registers a PV-WAVE input source handler procedure.

Usage

status = WtInput(function [, parameters])

Input Parameters

function:

'Add' — Add input to the source of events. If this function is specified, provide the following parameters:

- file_lun LUN of the input source file (or pipe).
- handler (optional) A PV-WAVE procedure that is called when input is available.
- client_data (optional) User data to be passed to the handler procedure.

'Remove' — Remove a previously registered input source. If this function is specified, provide the following parameter:

• input_id — ID of the input source being removed.

Returned Value

```
For 'Add':
```

status — The input source ID, or zero (0) to indicate failure.

For 'Remove':

status — One (1) indicates success; zero (0) indicates failure.

Keywords

If the 'Add' function is specified, the following keywords can be used:

Read — If specified and nonzero, the Xt Intrinsic condition XtInputReadMask is used to define the input source (default).

Write - If specified and nonzero, the Xt Intrinsic condition XtInputWriteMask is used to define the input.

Except — If specified and nonzero, the Xt Intrinsic condition XtInputExceptMask is used to define the input source.

Discussion

While most GUI applications are driven only by events, some applications need to incorporate other sources of input into the X Toolkit event handling mechanism. WtInput supports input or output gathering from files. The application registers an input source handler procedure and a file with the X Toolkit. When input is pending on the file, the registered handler is invoked.



In this context a "file" should be loosely interpreted to mean any sink (destination of output) or pipe (source of data).

For information on the requirements for writing input handler procedures, see Adding Input Handler Procedures on page 492 of the PV-WAVE Programmer's Guide.

Example

The following application accepts and processes data from another application that gathers the data.

```
PRO Server, top, client data, nparams, id, $
   lun, source
   READU, lun, data
       Process received data here.
END
PRO ButtonCB, wid, index
    COMMON ProcessComm, top, inputid
       Handle buttons here.
   CASE index OF
    1: ; Store
    2: ; Display
    3: BEGIN ; Close application
    status = WtInput('REMOVE', inputid)
```

```
status = WwSetValue(top, /Close)
    END
    ENDCASE
   END
PRO ProcessData
   COMMON ProcessComm, top, inputid
       Spawn data gathering application
   SPAWN, 'getdata', unit = lun
        Initialize the application.
   top = WwInit('processdata','Test', layout)
   buttons = WwButtonBox(layout, $
   ['Store','Display','Close'],'ButtonCB')
       Register input source handler and start the application.
   inputid = WtInput('ADD', lun, 'Server')
   status = WwSetValue(top, /Display)
   WwLoop
 FREE LUN, lun
END
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WtList Function

Controls the characteristics of scrolling list widgets.

Usage

status = WtList(function, widget [, parameters])

Parameters

function:

- 'Add' Add specified item(s) to the list.
- 'Delete' Delete specified item(s) from the list.
- 'DeleteAll' Delete all items from the list.
- 'Deselect' Deselect specified item(s) in the list.
- 'DeselectAll' Deselect all items in the list.
- 'ItemCount' Number of items in the list.
- 'Items' Get items in the list.
- 'Menu' (OLIT) Defines a popup menu for a scrolling list widget.
- 'Select' Select specified item(s) from the list.
- 'SelectAll' Select all items from the list.
- 'SelectCount' Number of selected items.
- 'Selected' Get selected items.
- 'Replace' Replace specified item(s) in the list.

widget — The list widget ID.

parameters:

For 'Add' $-(p_1)$ A string or array of strings; (p_2) a long integer representing the position at which to add the item. This optional parameter is used with Motif only. The default is the end position.

- For 'Delete' $-(p_1)$ A string or array of strings.
- For 'Deselect' $-(p_1)$ A string or array of strings.
- For 'ItemCount' $-(p_I)$ Number of items in the list (long, output).
- For 'Items' $-(p_I)$ Items in the list. Array of strings (output).
- For 'Menu' $-(p_I)$ Menu shell ID (long).
- For 'Select' $-(p_I)$ A string or array of strings.
- For 'SelectCount' $-(p_I)$ Number of selected items (long, output).
- For 'Selected' $-(p_1)$ Selected items. Array of strings (output).
- For 'Replace' $-(p_1)$ A string or array of strings; (p_2) position of first items to be replaced.

Keywords

Notify — If specified, a PV-WAVE callback is called during Select or Deselect operations.

Discussion

The list utility lets users choose an action from a list of actions. For detailed information on the list utility, see the description of XmList in the OSF/Motif Programmer's Reference. The scrolling list implemented for OLIT users is not the standard OLIT scrolling list widget. A nonstandard OLIT scrolling list was implemented for greater compatibility with the Motif scrolling list. The Widget Toolbox scrolling list for OLIT works much like the Motif scrolling list.

Example

```
.
.
items = ['Presidents Day','St.Patricks Day', $
```

```
'Easter', 'Memorial Day', '4th of July', $
   'Labor Day', 'Halloween', 'Thanksgiving', $
   'Hanukkah', 'Christmas', 'New Years Eve']
status = WtList("Add", list, items)
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WtLoop Function

Handles the dispatching of events and calling of PV-WAVE callback routines.

Usage

WtLoop

Parameters

None.

Returned Value

None.

Discussion

WtLoop causes PV-WAVE to loop indefinitely, processing the events and dispatching callbacks, handlers, and timers.

Motif Example

This example creates a Motif button labeled Done.

```
PRO example

COMMON block, top

@wtxmclasses.pro

top=WtInit('wt_ex4', 'Examples')

widget=WtCreate('Done', $

   xmPushButtonWidgetClass, top)

status=WtAddCallback(widget, $
   'activateCallback', 'CancelHelp')

status=WtSet(top, /Realize)

WtLoop

END
```

WtMainLoop Function

Handles the dispatching of events.

Usage

status = WtMainLoop

Input Parameters

None.

Output Parameters

None.

Returned Value

status — One (1) indicates success; zero (0) indicates failure.

Discussion

The WtLoop function can be called to accomplish the same result as WtMainLoop. WtMainLoop has been retained to provide upward compatibility with an earlier release of PV-WAVE, but it is recommended that you use WtLoop, instead.

See Also

WtLoop

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WtPointer Function

The pointer utility function.

Usage

status = WtPointer(function, widget [, parameters])

Parameters

function:

- 'GetLocation' Get the current location of the pointer.
- 'GetControl' Get the control attributes of the pointer.
- 'GetMapping' Get the current mapping of the pointer.
- 'SetLocation' Set the location of the pointer.
- 'SetControl' Set the control attributes of the pointer.
- 'SetMapping' Set the mapping of the pointer.

widget - The current widget.

parameters:

- 'GetLocation' A seven-element array of long integers:
 - 0 Root window
 - 1 Child window where the pointer is located
 - 2 X coordinate in root
 - 3 Y coordinate in root
 - 4 X coordinate in current window
 - 5 Y coordinate in current window
 - 6 Modifier keys and buttons state mask
- 'GetControl' A three-element array of long integers:
 - 0 accel_numerator

- 1 accel denominator
- 2 threshold
- 'GetMapping' An array of up to ten long integers:
 - 0–9 Pointer button mapping
- ${\tt 'SetLocation'}-A$ two element array of long integers:
 - 0 X coordinate in current window
 - 1 Y coordinate in current window
- 'SetControl' A three-element array of long integers:
 - 0 accel_numerator
 - 1 accel denominator
 - 2 threshold
- 'SetMapping' An array of up to ten long integers:
 - 0-9 Pointer button mapping

Returned Value

status — One (1) indicates success; zero (0) indicates failure.

Discussion

See the Xlib Reference Manual (O'Reilly & Associates, Inc., 1989) for details on XQueryPointer, XGetPointerControl, XGetPointerMapping, XWarpPointer, XChangePointerControl, and XSetPointerMapping.

Example

```
status=WtPointer("GetLocation", wid, state)
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, *Using the Widget Toolbox*, in the *PV-WAVE Programmer's Guide*.

WtSet Function

Sets widget resources.

Usage

status = WtSet(widget, argv)

Parameters

widget — The widget ID.

argv — A PV-WAVE unnamed structure specifying the resources for the widget or shell.

Returned Value

status — One (1) indicates success; zero (0) indicates failure.

Keywords

Append=com — (Motif only) Appends a command to the command widget.

Callback=reason — Calls all defined callbacks for the specified reason for this widget.

Destroy - Destroys the widget and all children of the widget.

Error=errmsg — (Motif only) Displays an error message in the history area of the command widget.

Manage — Manages the given widget (WtCreate manages widgets by default).

Map - Maps the given widget.

Mappedwhen — Sets mapped to true for managed attributes.

Nonsensitive — Sets the given widget to nonsensitive.

Popup=event – Pops up the given menu (event specifies the location).

Realize — Realizes (displays) the given widget.

Realize=grab — Realizes the given shell with one of these grab values:

Xtgrabnone

Xtgrabnonexclusive

Xtgrabexclusive

 $Remove_callback = name - If the name of a PV-WAVE callback$ routine is specified, this keyword removes the named callback. If name is not specified, all callbacks for the specified widget are removed.

 $Remove_handler = name - If the name of a PV-WAVE event$ handler is specified, this keyword removes the named event handler. If name is not specified, all event handlers for the specified widget are removed.

Search=dir — (Motif only) Sets the search context for a file selection widget.

Sensitive — Sets the given widget to sensitive.

Unmanage — Unmanages the given widget.

Unmap — Unmaps the given widget.

Unmappedwhen — Sets mapped to false for managed attributes.

Unrealize — If present and nonzero, unrealizes a widget, or pops down (undisplays) a shell.

Userdata — Stores a copy of the value of a PV-WAVE variable.

Value=value – (Motif only) Sets the value for a command, scale, scroll bar, or toggle button.

Discussion

Resources are passed as an unnamed structure, where tag names correspond to resource names and tag definitions are resource values. Resource names are given without the XtN (OLIT) or XmN (Motif) prefix.

An unnamed structure has the following general definition:

```
x = \{, tag\_name_1: tag\_def_1, tag\_name_n: tag\_def_n\}
```

See the *Creating Unnamed Structures* on page 105 of the *PV-WAVE Programmer's Guide* for detailed information on unnamed structures.

For a list of resources for the specified widget or shell, see the appropriate OSF/Motif Programmer's Reference or the OLIT Programmer's Reference.



If the color, font, or bitmap (called an "image" in OLIT and a "pixmap" in Motif) resource value is of type string, it is assumed to be the value of the color/font name or bitmap file name, and the appropriate resources are loaded.

Example

```
.
.
args={,x:100,y:100,label:'Enter File:',$
    string:'/usr/home/myfile.pro'}
    Resources are defined in an unnamed structure.
status=WtSet(w,args)
.
.
status=WtSet(top,/Realize)
.
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WtTable Function

Modifies an xbaeMatrix class widget.

Usage

status = WtTable(function, widget [, parameters])

Input Parameters

function:

- 'AddColumns' Add columns to the table. If this function is specified, also provide the following parameters:
- columns Two-dimensional string array of column values.
- num_columns Number of columns to be added.
- width One-dimensional array of column widths.
- labels (optional) One-dimensional array of column labels.
- max lengths (optional) One-dimensional array of column maximum lengths.
- alignments (optional) One-dimensional array of column alignments. Valid values are:
 - 0 Align cell contents to cell's left edge (left justify).
 - 1 Center cell contents (center justify).
 - 2 Align cell contents to cell's right edge (right justify).

- label_alignments (optional) One-dimensional array of column label alignments. Valid values are:
 - 0 Align cell contents to cell's left edge (left justify).
 - 1 Center cell contents (center justify).
 - 2 Align cell contents to cell's right edge (right justify).
- colors (optional) Two-dimensional array of column cell colors.
- 'AddRows' Add rows to the table. If this function is specified, also provide the following parameters:
- rows Two-dimensional string array of row values.
- num rows Number of rows to be added.
- labels (optional) One-dimensional array of row labels.
- colors (optional) Two-dimensional array of row cell colors.
- 'DeleteCols' Delete columns from the table. If this function is specified, also provide the following parameter:
- num columns Number of columns to delete.
- 'DeleteRows' Delete rows from the table. If this function is specified, also provide the following parameter:
- num_rows Number of rows to delete.
- 'DeselectAll' Deselect all cells in the table.
- 'DeselectCell' Deselect the specified cell in the table. If this function is specified, also provide the following parameter:
- row Row index of the cell to deselect.
- column Column index of the cell to deselect.
- 'DeselectCol' Deselect the specified column in the table. If this function is specified, also provide the following parameter:
- column Index of column to deselect.
- 'DeselectRow' Deselect the specified row in the table. If this function is specified, also provide the following parameter:
- row Index of row to deselect.

- 'SelectCell' Select the specified cell in the table. If this function is specified, also provide the following parameters:
- row Row of the cell to select.
- column Column of the cell to select.
- 'SelectCol' Select the specified column in the table. If this function is specified, also provide the following parameter:
- column Column to select.
- 'SelectRow' Select the specified row in the table. If this function is specified, also provide the following parameter:
- row Row to deselect.
- 'EditCell' Edit the specified cell in the table. If this function is specified, also provide the following parameters:
- row Row of the cell to edit.
- column Column of the cell to edit.
- 'CancelEdit' Cancel the edit of the currently edited cell. If this function is specified, also provide the following parameter:
- unmap (optional) If specified and nonzero, the currently edited cell is unmapped.
- 'Committedit' Commit the edit of the currently edited cell. If this function is specified, also provide the following parameter:
- unmap (optional) If specified and nonzero, the currently edited cell is unmapped.
- 'GetCell' Get the value of a cell in the table. If this function is specified, also provide the following parameters:
- row Row of the cell to get.
- column Column of the cell to get. Returns the value of the specified cell.
- 'SetCell' Set the value of a cell in the table. If this function is specified, also provide the following parameters:
- row Row of the cell to set.
- column Column of the cell to set.

- value New value of the specified cell.
- 'SetColor' Set the color value for a cell in the table. If this function is specified, also provide the following parameters:
- row Row of the cell whose color you want to set.
- column Column of the cell whose color you want to set.
- color New colormap index of the specified cell.
- 'SetColColor' Set the color value for a column in the table. If this function is specified, also provide the following parameters:
- position Column at which to start setting colors.
- colors Two-dimensional array of new colormap indexes for columns.
- num colors The number of elements in the colors array.
- 'SetRowColor' Set the color value for a row in the table. If the 'SetRowColor' function is specified, also provide the following parameters:
- position Row at which to start setting colors.
- colors 2D array of new colortable indexes for rows.
- num_colors The number of colors in the colors array.

widget - Widget ID of the table (xbaeMatrix class) widget.

Discussion

The XbaeMatrix widget creates an editable 2D array of string data (cells) similar to a spreadsheet. All values must be converted to string type before being set with WtTable.

XbaeMatrix Widget Documentation

The Motif version of the XbaeMatrix widget was originally developed by Andrew Wason of Bellcore. This widget was enhanced and an OLIT version was developed by Visual Numerics, Inc. Complete documentation for the XbaeMatrix widget is available in \$WAVE DIR/docs/widgets:

- $matrix_motif.ps A PostScript file containing the$ documentation for the Motif version of XbaeMatrix widget. You can print this file on any PostScript printer.
- matrix olit.ps A PostScript file containing the documentation for the OLIT version of XbaeMatrix widget. You can print this file on any PostScript printer.

XbaeMatrix Widget Callbacks

See Appendix C, Motif and OLIT Callback Parameters in the PV-WAVE Programmer's Guide for information on the required parameters for all PV-WAVE widget callbacks. In addition, the XbaeMatrix widget's callback routines and their parameters are documented in the files matrix motif.ps and matrix olit.ps discussed in the previous section. Refer to those documents for information on the table widget's callbacks.

Example

In this example, values in the table are modified (set) through a call to WtTable.

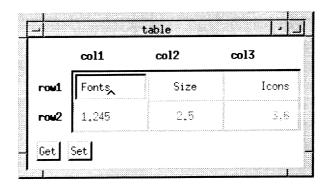


Figure 2-82 Editable table based on the XbaeMatrix widget.

PRO ButtonCB, w, which COMMON Widgets, table

When a button is pressed, modify the table with a call to WtTable.

```
1: PRINT, WtGet(table, 'cells', Nrows = 2, $
        Ncols = 3)
     2: status = WtTable('SetCell', table, 0, $
        0, 'New Font')
ENDCASE
        END
        PRO LeaveCellCB, w, data, n, reason, event, r,
           c, $
           value, doit
    This callback is called just before an edit to a cell is committed. For more
    information on XbaeMatrix widget callbacks and their parameters, see the
    previous section "XbaeMatrix Widget Callbacks".
 PRINT, "leaveCellCallback ", r, c, value
       END
       PRO ModifyVerifyCB, w, data, n,reason, event,$
           r, c, doit, ci, cn, sp, ep, txt, len, fmt
   This callback is called while a cell is being edited. For more information
   on XbaeMatrix widget callbacks and their parameters, see the previous
   section "XbaeMatrix Widget Callbacks".
        print, "modifyVerifyCallback ", r, c, doit,
           ci, cn, $ sp, ep, txt, len, fmt
       END
       PRO Table
       @wtxmclasses
       @wtxmconsts
       COMMON Widgets, table
    Initialize the Widgets Toolkit.
 top = WwInit('table','Test', layout, $
   /Vertical)
   Load color table.
loadct, 5
   Set cell contents, display table.
```

CASE which OF

```
cells = [['Fonts','Size','Icons'], $
 ['1.245','2.5','3.6']]
 args = {, rows:2, columns:3, cells:cells,$
  columnWidths:[10, 10, 10], $
columnLabels:['col1','col2','col3'],$
   rowLabels:['row1','row2'], $
  boldLabels:TRUE, $
  colors:[[40, 50, 60], [100, 110, 120]], $
columnAlignments: [XmALIGNMENT BEGINNING, $
  Xmalignment center, Xmalignment_end]}
table = WtCreate('table', xbaeMatrixWidgetClass,
   layout, args)
status = WtAddCallback(table, "leaveCellCallback",$
   'leaveCellCB')
      status = WtAddCallback(table, "modifyVerify-
         Callback",$ 'modifyVerifyCB')
     buttons = WwButtonBox(layout,['Get','Set'],'-
         ButtonCB')
     status = WwSetValue(top, /Display)
     WwLoop
     END
```

See Also

WwTable

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WtTimer Function

Registers a callback function for a given timer.

Usage

status = WtTimer(function, params, [client_data])

Parameters

function:

 'Add' — If Add is specified, the also provide the following three parameters:

time — A long integer specifying the time interval in milliseconds.

timer — A string containing a PV-WAVE timer function name.

client_data — (optional) A PV-WAVE variable.

 'Remove' — If Remove is specified, also provide the following parameter:

ID — The timer ID.

Returned Value

status — For Add, returns the timer ID or zero (0) to indicate failure. For Remove, one (1) indicates success; zero (0) indicates failure.

Keywords

Once_only — If present and nonzero, the PV-WAVE timer function is called only once.

Discussion

This timer, unlike the XtIntrinsics XtTimeOut function, restarts itself. To stop the timer, use the command:

```
WtTimer("REMOVE", id)
```

in the timer callback. The new timer ID is returned in the timer callback parameter timer id.

If you need to keep a copy of the timer ID in a common block, copy the timer id parameter of the timer callback routine to your timer ID variable in the COMMON block, as shown in the following example.

For information on the requirements for writing timer procedures, see Adding Timers on page 489 of the PV-WAVE Programmer's Guide.

Example

```
COMMON timer, tid
id=WtTimer("ADD", 100, 'TimerCallback', $
  my data)
PRO TimerCallback, wid, client data, $
   timer id, interval
   COMMON timer, tid
   tid = timer_id
END
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WtWorkProc Function

Registers a PV-WAVE work procedure for background processing.

Usage

status = WtWorkProc(function, parameters)

Parameters

function:

- 'Add' Registers the specified work procedure.
- 'Remove' Removes the previously registered work procedure.

parameters:

If 'Add' is specified, also provide the following two parameters:

```
workproc - A PV-WAVE work procedure name (string).client_data - (optional) A PV-WAVE variable.
```

If 'Remove' is specified, also provide the following parameter:

id — The ID of the work procedure to be removed.

Returned Value

```
For 'Add':

status — The work procedure ID; zero (0) indicates failure.

For 'Remove':

status — One (1) indicates success; zero (0) indicates failure.
```

Discussion

A work procedure is useful if you need to execute a timeconsuming operation from a callback procedure. While a callback procedure normally blocks events until the operation is finished, a work procedure lets you execute a callback while the toolkit is idle.

For information on the requirements for writing work procedures, see Adding Work Procedures on page 491 of the PV-WAVE Programmer's Guide.

Example

PRO ButtonCB

This callback is called when a button is selected. Do not initiate the selected operation here, because it is too time-consuming. Instead, schedule a work procedure.

```
id=WtWorkProc("ADD", 'MyWorkProc', my data)
END
PRO MyWorkProc, wid, client data, workproc id
IF done THEN status = WtWorkProc("REMOVE", $
  workproc id)
END
```

See Also

For more information about how to write an application program based on PV-WAVE's Widget Toolbox, refer to Chapter 16, Using the Widget Toolbox, in the PV-WAVE Programmer's Guide.

WwButtonBox Function

Creates a horizontally or vertically oriented box containing push buttons.

Usage

bbox = WwButtonBox(parent, labels, callback)

Input Parameters

parent — The parent widget's ID.

labels - A string or an array of strings containing the text that is to appear on the buttons.

callback — A string containing the name of the PV-WAVE callback routine.

Returned Value

bbox — The ID of the button box widget.

Input Keywords

Border — Specifies the width in pixels of the button box and button borders.

Horizontal — If present and nonzero, creates a horizontally aligned row of buttons.

Measure — Specifies the number of columns of buttons (for a vertical box) or rows (for a horizontal box).

Position — If the button box widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the button box widget within the bulletin board.

Vertical — When present and nonzero, creates a vertically aligned column of buttons.

Spacing — Specifies the space in pixels between buttons.

Output Keywords

Buttons — Returns an array of push button widget IDs.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background - Specifies the background color name. Background color is the color of the button.

Basecolor - Specifies the base color of the container widget.

Font — Specifies the name of the font used for button text.

Foreground — Specifies the foreground color name. Foreground color is the color of the button text.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom - If a widget ID is specified (for example, Bottom= wid), then the bottom of the button box widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the button box widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the button box widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the button box widget is attached to the left side of the parent widget.

Right - If a widget ID is specified (for example, Right=wid),then the right side of the button box widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the button box widget is attached to the right side of the parent widget.

Top - If a widget ID is specified (for example, Top=wid), then the top of the button box widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the button box widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets the label of the selected button.

setvalue — Sets the label of the selected button.

Callback Parameters

Any button box callback procedure must have the following two parameters:

wid — The button widget ID.

index — Index of the button pushed (1 - n).

Discussion

The "button box" is a special widget in which individual buttons are arranged. If only one button is requested, that button's widget ID is returned. By default, the buttons are arranged in a row/column format. See the WwLayout function for information on row/ column format.

Example

This example creates a button box containing three buttons, Quit, Dialog, and Message. The callback ButtonCB is executed when one of the buttons in the button box is selected.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the button box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

```
PRO ButtonCB, wid, data
CASE data OF
   1: PRINT, 'Quit Selected'
   2: PRINT, 'Dialog Selected'
   3: PRINT, 'Message Selected'
   ENDCASE
END
```

Widget Commands

```
labels = ['Quit','Dialog','Message']
top=WwInit('ww ex20', 'Examples', layout)
bbox=WwButtonBox(layout, labels, 'ButtonCB', $
   /Horizontal, Spacing=20)
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwCommand Function

Creates a command window.

Usage

Input Parameters

parent — The parent widget's ID.

enteredCallback — A string containing the name of the PV-WAVE callback routine that is executed when a command is entered and confirmed (the user presses <Return>).

doneCallback — A string containing the name of the callback routine that is executed when the command window is destroyed.

Returned Value

command — The widget ID of the command window.

Keywords

Maximum — Specifies the maximum number of items that can be placed in the command history list.

Position — Specifies the position of the upper-left corner of the command window on the screen.

Prompt — Specifies a string containing the command prompt.

Title - Specifies a string containing the command window title.

Visible — Specifies the maximum number of command history items that are visible.

Width – Specifies the width of the command window.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font — Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets a string array containing the list of commands entered, from the oldest to newest command.

setvalue — Sets a new command in the text input field.

Callback Parameters

Any command widget callback procedure must have the following two parameters:

container - Container widget ID.

wid - Popup window widget ID.

Discussion

A command window is a *popup* window. This means that it cannot be the child of the top-level shell or the layout widget. Usually, a command widget is activated by a pushbutton or menu button, as in the example below.

A command window provides:

- A text input field where the user can enter text, such as a command or a label.
- A scrolling command-history list. Whenever the user enters a
 command in the text input field and presses <Return>, a callback is executed and the command text is placed on the history
 list. The user can re-enter a previously entered command by
 clicking on it in the command history list.

If the user double-clicks on a command in the command history list, the callback is automatically executed and appended to the end of the list.

• A label for the text input field.

Example

This example creates a button labeled Command. When this button is selected, the CbuttonCB callback is activated and a command window is created. When a user enters text in the text input field and presses <Return>, the callback CommandOK is executed. When the user exits the command box, the CommandDone callback is executed.

Enter the callback procedures into a file, and compile the procedures with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the widgets, select the appropriate function (such as Close) from the window manager menu of the Command button (the parent widget).

Callback Procedures

```
PRO CbuttonCB, wid, data
   command = WwCommand(wid, 'CommandOK', $
      'CommandDone', Position=[300,300], $
      Title='Command Entry Window')
END
PRO CommandOK, wid, shell
   value = WwGetValue(wid)
   PRINT, value
END
PRO CommandDone, wid, shell
   status = WwSetValue(shell, /Close)
END
```

Widget Commands

```
top=WwInit('example2', 'Examples', layout)
button=WwButtonBox(layout, 'Command', $
   'CbuttonCB')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwControlsBox Function

Creates a box containing sliders.

Usage

Input Parameters

parent — The widget ID of the parent widget.

labels — A string or array of strings containing the text used to label the slider ranges.

range — An array of values specifying the minimum and maximum slider values.

changedCallback — A string containing the name of the callback routine that is executed when the value of a slider changes.

Returned Value

controls — The widget ID of the controls box widget.

Input Keywords

Border — Specifies the width in pixels of the controls box and slider borders.

Drag — If present and nonzero, the callback procedure is called while the slider is being dragged.

Float — Lets you display a floating-point slider, with ranges that include implied decimal numbers. This keyword specifies the number of digits that appear after the decimal point. For example, to display a floating-point slider with the range 0.5 to 12.5, use:

```
controls=WwControlsBox(parent, 'myslider',$
  [5, 125], 'mycallback', Float=1)
```

The decimal point is implied in that it is used for display purposes only. If the user choses a slider value of 5.7, the value returned to the callback is 57. The value of *Float* can be obtained through the Userdata keyword of the WwGetValue function. Thus, you can obtain the implied decimal position that was used in a WwControlsBox function. For example:

```
PRO mycallback, wid, which
   ndec = WwGetValue(wid, /Userdata)
  value = WwGetValue(wid)
  PRINT, 'User Selected Value: ', $
  Float(value)/10.0^ndec
```

Height – Specifies the height of the slider(s).

Horizontal — When present and nonzero, creates a horizontally aligned row of sliders.

Hslider - When present and nonzero, creates horizontallyoriented slider(s).

Measure - Specifies the number of columns of sliders (for a vertical box) or rows (for a horizontal box).

Position — If the controls box widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the controls box widget within the bulletin board.

Spacing — Specifies the space between sliders.

Text — When present and nonzero, an input text field is created for each slider.

Vertical — When present and nonzero, creates a vertically aligned column of sliders.

Vslider — When present and nonzero, creates vertically-oriented slider(s).

Width — Specifies the width of the slider(s).

Output Keywords

Sliders — Returns an array of slider widget IDs.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font - Specifies the name of the font used for slider text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the controls box widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the controls box widget is attached to the bottom of the parent widget.

Left – If a widget ID is specified (for example, Left=wid), then the left side of the controls box widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the controls box widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the controls box widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the controls box widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the controls box widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the controls box widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets the value of the selected slider.

setvalue — Sets the value of the selected slider.

Callback Parameters

Any controls box callback procedure must have the following two parameters:

wid — Slider widget ID.

index — Index value of the slider that has changed (1 - n).

Discussion

A "controls box" is a special widget in which individual sliders are arranged. If only one slider is requested, that slider's widget ID is returned. By default, the sliders are placed in a row/column format.

A slider allows the user to change a value interactively by moving the slider handle back and forth within a predefined range. You have the option of including a text input field with each slider. The text input field lets the user enter an exact value for the slider.

Example

This example creates a box with three sliders labeled Pressure, RPM, and Temperature. Whenever the user moves one of the sliders, the callback procedure is executed.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the controls box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

```
PRO SliderCB, wid, which

CASE which OF

1: PRINT, 'First Slider Moved'

2: PRINT, 'Second Slider Moved'

3: PRINT, 'Third Slider Moved'

ENDCASE

value = WwGetValue(wid)

PRINT, value

END
```

Widget Commands

```
top=WwInit('ww_ex22', 'Examples', layout)
labels=['Pressure','RPM','Temperature']
ranges=[0,100,2000,4000,50,150]
controls = WwControlsBox(layout, labels, $
    ranges, 'SliderCB',/Vertical,/Text, $
    Foreground='gray',Background='white', $
    Basecolor='blue')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwDialog Function

Creates a blocking or nonblocking dialog box.

Usage

wid = WwDialog(parent, label, OKCallback, CancelCallback, HelpCallback)

Input Parameters

parent — The widget ID of the parent widget.

label — A string containing the label for the input field.

OKCallback - A string containing the name of the callback that is executed when the OK (Motif) or Apply (OPEN LOOK) button is selected.

CancelCallback — A string containing the name of the callback that is executed when the Cancel button is selected.

HelpCallback - A string containing the name of the callback that is executed when the Help button is selected. If you are running under OPEN LOOK, this parameter is ignored.

Returned Value

wid — The ID of the dialog widget.

Keywords

Block — If this keyword is present and nonzero, the dialog box is blocking (the default).

Cols – Specifies the number of columns in the text input field.

Nonblock — If this keyword is present an nonzero, the dialog box is not blocking.

Text — Specifies a string containing the initial text in the text input field.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets a string containing the text entered in the text input field.

setvalue — Sets a string in the text input field.

Callback Parameters

Any dialog box callback procedure must have the following two parameters:

wid - Command widget ID.

text - Text input field widget ID.

Discussion

A dialog box is a *popup* window. This means that it cannot be the child of the top-level shell or the layout widget. Usually, a dialog box widget is activated by a pushbutton or menu button, as in the example below.

Example

This example creates a button labeled Dialog Box. When the user selects this button, a dialog box appears. When the user enters text in the dialog box and presses <Return>, DialogOK is executed. When the user cancels the dialog box, the second callback routine, DialogCancel, is executed.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss both widgets, select the appropriate function (such as Close) from the window manager menu of the Dialog Box button (the parent widget).

Callback Procedures

```
PRO DbuttonCB, wid, data
   select=WwDialog(wid, 'Type something:',$
      'DialogOK', 'DialogCancel', Title='Type')
END
PRO DialogOK, wid, text
   PRINT, 'Dialog OK'
   value = WwGetValue(text)
   PRINT, value
END
PRO DialogCancel, wid, data
   PRINT, 'Dialog Cancel'
END
```

Widget Commands

```
top=WwInit('ww ex23', 'Examples', layout)
button=WwButtonBox(layout, 'Dialog Box', $
   'DbuttonCB')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwDrawing Function

Creates a drawing area, which allows users to display graphics generated by PV-WAVE.

Usage

Input Parameters

parent - The widget ID of the parent widget.

windowid — The window ID of the PV-WAVE graphics window. When the window index is undefined or between zero and 31, the first free window index (ascending) is used and returned as windowid. (For information on window IDs, see the WINDOW procedure.)

drawCallback — A string containing the name of the PV-WAVE callback routine that is executed when the drawing area is exposed to display the graphics.

wsize — A vector containing two long integers that represent the width and height of the drawing area window.

dsize — A vector containing two long integers that represent the width and height of the image to be displayed in the drawing area. If this is larger than wsize, you can use the scroll bars to move the image around in the display window.

Returned Value

wid — The ID of the drawing area widget.

Input Keywords

Border — Specifies the width of the borders in pixels for the parent widget and the child widgets. The default is 0.

Noscroll - If present and nonzero, the drawing area does not use scroll bars. The drawing area window is created the same size as the drawing area. In other words, the value of dsize equals wsize.

Position — If the drawing box widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the drawing box widget within the bulletin board.

Output Keywords

Area – Returns the drawing area widget's ID.

Color Keywords

For additional information on the color keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the drawing box widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the drawing box widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the drawing box widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the drawing box widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the drawing box widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the drawing box widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the drawing box widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the drawing box widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets the window ID of the drawing area widget.

setvalue — Sets a two-element vector containing the width and height of the drawing area.

Callback Parameters

Any drawing area widget callback procedure must have the following two parameters:

wid - Drawing area widget ID.

index - PV-WAVE window index.

Example

This example creates a widget that displays a PV-WAVE image. Whenever the drawing area widget is displayed, the callback is executed. In this case, the callback opens and reads an image file.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the drawing area widget, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

```
PRO DrawCB, wid, data
   common draw, img
   print, 'Draw'
   tv, img
END
```

Widget Commands

```
top=WwInit('ww_ex24', 'Examples', layout)
COMMON draw, imq
LOADCT, 5, /SILENT
img=BYTARR(512,512)
OPENR, 1, '$WAVE DIR/data/head.img'
   Note: Under VMS, enter: WAVE_DIR:[DATA]head.img
READU, 1, img
CLOSE, 1
draw=WwDrawing(layout, 1,'DrawCB', $
   [256,256], [512,512])
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwFileSelection Function

Creates a file selection widget, which lets the user display the contents of directories and select files.

Usage

wid = WwFileSelection(parent, OKCallback, CancelCallback, HelpCallback)

Input Parameters

parent — The widget ID of the parent widget.

OKCallback — A string containing the name of the callback that is executed when the OK button is selected.

CancelCallback — A string containing the name of the callback procedure that is called when the file selection widget is dismissed.

HelpCallback — A string containing the name of the callback routine that is called when the Help button is selected. If you are running under OPEN LOOK, this parameter is ignored.

Returned Value

wid — The file selection widget ID.

Keywords

Block – Creates a blocking file selection window (the default).

Dir – Specifies a string containing the directory path.

File – Specifies a string containing the default file selection.

NonBlock — Creates a nonblocking file selection window.

Pattern — (Motif only) Specifies the search pattern used in combination with the directory in determining files to be displayed.

Position — Specifies the position of the upper-left corner of the file selection window on the screen in pixels.

Title - Specifies a string containing the file selection widget's title.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font — Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets the selected file specification.

setvalue — Sets a three-element array of strings:

- 0 Determines the files and directories displayed in the directory list. For example, /usr/home/mydir/*.c. Ignored in OLIT.
- 1 Directory: specifies the base directory.
- 2 Pattern: specifies the search pattern to be used to select files.

Callback Parameters

Any file selection widget callback procedure must have the following two parameters:

wid - File selection widget ID.

shell — The ID of the top-level shell.

Discussion

A file selection widget is a *popup* window. This means that it cannot be the child of the top-level shell or the layout widget. Usually, a file selection widget is activated by a pushbutton or menu button, as in the example below.

Example

This example creates a button labeled File Selection. When the user selects this button, a file selection widget appears. When the user selects a file, the callback is executed.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the widget, select the appropriate function (such as Close) from the window manager menu of the File Selection button (the parent widget).

Callback Procedure

```
PRO FbuttonCB, wid, data
   file = WwFileSelection(wid, 'FileOK',$
      'FileCancel', Title='Search')
END
PRO FileOK, wid, shell
   value = WwGetValue(wid)
   PRINT, value
   status = WwSetValue(shell, /Close)
END
PRO FileCancel, wid, shell
   PRINT, 'File Cancel'
END
```

Widget Commands

```
top=WwInit('ww_ex25', 'Examples', layout)
button=WwButtonBox(layout, 'File Tool', $
   'FbuttonCB')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwGetValue Function

Returns a specific value for a given widget.

Usage

value = WwGetValue(widget)

Input Parameters

widget - The widget for which you want the value.

Returned Value

value — The value returned from the widget.

Keywords

Sensitive — Returns 1 if the widget is sensitive, or 0 if the widget is not sensitive.

Shown — Returns 1 if the widget is shown, or 0 if the widget is not visible.

Userdata — Returns the value of the *Userdata* variable that was previously stored with the WwSetValue function.

Discussion

See the *Get Value* section under each WAVE Widget function description to find out what WwGetValue returns by default for each function. For example, WwGetValue called with the ID of a list widget returns a string array containing the selected item(s) in the list.

Example

The following example demonstrates WwGetValue with the WwCommand function. WwGetValue returns a string array containing the commands entered in the Command window. The

callback routine CommandOK prints the value returned by WwGetValue whenever the user enters a command in the Command window and presses < Return>.

Enter the callback procedures into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the widget, select the appropriate function (such as Close) from the window manager menu of the command window.

Callback Procedures

```
PRO CbuttonCB, wid, data
   command = WwCommand(wid, 'CommandOK', $
      'CommandDone', Position=[300,300], $
      Title='Command Entry Window')
END
PRO CommandOK, wid, shell
   value = WwGetValue(wid)
   print, value
END
PRO CommandDone, wid, shell
   status = WwSetValue(shell, /Close)
END
```

Widget Commands

```
top=WwInit('ww ex26', 'Examples', layout)
button=WwButtonBox(layout, 'Command', $
   'CbuttonCB')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwInit Function

Initializes the WAVE Widgets environment, opens the display, creates the first top-level shell, and creates a layout widget.

Usage

topshell = WwInit(app_name, appclass_name, workarea
[, destroyCallback])

Input Parameters

app_name — A string containing the name of the application.
 This name can be referenced in a resource file.

appclass_name — A string containing the application class name, which can be the name of a resource file.

destroyCallback — (optional) A string containing the name of the callback that is executed when the top-level shell is destroyed.

Output Parameters

workarea — The widget ID of the layout widget that is created inside of the top-level shell.

Returned Value

topshell — The widget ID of the top-level application shell.

Keywords

Board — If present and nonzero, a bulletin board layout is created inside the top-level shell.

Border — Specifies the width in pixels of the borders for the layout widget and its child widgets. Default is 0.

Form — If present and nonzero, a form layout is created inside the top-level shell.

Horizontal - If present and nonzero, orients child widgets horizontally within the layout widget (the default). Used with row/ column layouts only. For more information on layout widgets, see the WwLayout function.

Position – Specifies the position of the upper-left corner of the main window on the screen.

Spacing — Specifies the amount of space between child widgets inside the layout. Used with row/column layouts only. Default is 0. For more information on layout widgets, see the WwLayout function.

Title – A string specifying a title for the shell.

Vertical — If present and nonzero, orients child widgets vertically within the layout widget. Used with row/column layouts only. For more information on layout widgets, see the WwLayout function.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background - Specifies the default background color name for an application.

Font - Specifies the name of the default font used for text in an application.

Foreground — Specifies the default foreground color name for an application.

Discussion

Call this routine before the first use of a WAVE Widgets routine.

Example

For examples showing the use of WwInit, see any of the WAVE Widgets widget-creation routines, such as WwButtonBox, WwCommand, WwControlsBox, WwList, and so on.

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwLayout Function

Creates a layout widget that is used to control the arrangement of other widgets.

Usage

layout = WwLayout(parent)

Input Parameters

parent - The widget ID of the parent widget.

Returned Value

layout — The widget ID of the layout widget.

Keywords

Board — If present and nonzero, the layout that is created is a "bulletin board".

Border — Specifies the width of the borders in pixels for the parent widget and the child widgets. The default is 0.

Form — If present and nonzero, the layout that is created is a "form".

Horizontal - If present and nonzero, aligns child widgets horizontally within the layout widget (the default). This keyword is only used for row/column layouts.

Position — If the layout widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the layout widget within the bulletin board.

Scroll — If present and nonzero, places scroll bars on the layout widget. Specify a width and height (e.g., Scroll=[w,h]) to set the width and height of the scrolled window.

Spacing — Specifies the amount of space in pixels between child widgets in the layout. The default is 0. This keyword is only used for row/column layouts.

Vertical — If present and nonzero, aligns child widgets vertically within the parent widget. This keyword is only used for row/column layouts.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the layout widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the layout widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the layout widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the layout widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the layout widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the layout widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the layout widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the layout widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets the x, y position of the contents of the scrolled window.

setvalue — Sets the x, y position of the contents of the scrolled window.

Callback Parameters

None.

Discussion

A layout widget is a container that holds other widgets. The layout widget provides different methods of arranging widgets, such as buttons, menus, sliders, and even other layout widgets, inside the "container." The three layout types are:

 Row/column — Widgets are arranged in rows and/or columns (the default).

- Form Widgets are "attached" to one another inside the layout. Specified with the Form keyword.
- Bulletin board Widgets are positioned in the layout with x, y coordinates. Specified with the *Board* keyword.

The layout widget can be the parent widget for each "child" widget it contains. If a border is not specified with the Border keyword, the layout widget itself is not visible to the user; only the widgets inside the layout are visible.

For more detailed information on layouts, see Arranging Widgets in a Layout on page 419 of the PV-WAVE Programmer's Guide.

Examples

The following example creates a simple layout widget containing a button box and a radio box widget. Note that WwLayout does not use callbacks, as it primarily creates a "container" that holds other widgets.

Enter the callback procedures into a file, and compile them with the .RUN command. Then, enter the widget commands at the WAVE> prompt (or enter them in a command file and run them with the @ command). To dismiss the layout box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedures

```
PRO RadioCB, wid, which
CASE which OF
   1: PRINT, 'First Toggle Selected'
   2: PRINT, 'Second Toggle Selected'
   3: PRINT, 'Third Toggle Selected'
   ENDCASE
value = WwGetValue(wid)
PRINT, value
END
PRO ButtonCB, wid, data
CASE data OF
```

```
    PRINT, 'Quit Selected'
    PRINT, 'Dialog Selected'
    PRINT, 'Message Selected'
    ENDCASE
```

Widget Commands

```
top=WwInit('ww_ex27', 'Examples', layout,$
    /Vertical, Spacing=30, Border=10)

blabels = ['Quit','Dialog','Message']
bbox=WwButtonBox(layout, blabels,'ButtonCB',$
    /Horizontal, Spacing=20)

rlabels=['System','Owner','Group']
rbox=WwRadioBox(layout,rlabels, 'RadioCB', $
    /Vertical, Border=2, Spacing=20, $
    Top=controls)

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

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwList Function

Creates a scrolling list widget.

Usage

list = WwList(parent, items, selectedCallback, defaultCallback)

Input Parameters

parent - The widget ID of the parent widget.

items — A string array containing the items to appear on the list.

selectedCallback — A string containing the name of the callback that is executed when an item is selected.

defaultCallback — A string containing the name of the callback that is executed when a user double-clicks on an item.

Returned Value

list — The widget ID of the list widget.

Keywords

Multi — If present and nonzero, the list widget uses multiple selection mode. The default is single selection mode.

Position — If the scrolling list widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the scrolling list widget within the bulletin board.

Visible - Specifies the number of items that are visible in the list widget.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the scrolling list widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the scrolling list widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the scrolling list widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the scrolling list widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the scrolling list widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the scrolling list widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the scrolling list widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the scrolling list widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Returns an array of the selected items (string).

setvalue - Replaces current items with the new items (array of strings).

Callback Parameters

Any list widget callback procedure must have the following two parameters:

wid - List widget ID.

parent — Parent widget ID.

Discussion

A scrolling list allows users to select one or more items from a group of choices. Items are selected from the list with the mouse. An additional callback can be defined for the default action. This callback is executed when the user double-clicks on an item.

Example

This example creates a scrolling list containing the items defined in the string array items. When the user selects an item (clicks it with the mouse), the first callback ListCB is executed. If the user double-clicks on an item, the callback DefaultCB is executed.

Enter the callback procedures into a file, and compile them with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the list box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedures

PRO ListCB, wid, data PRINT, 'Item Selected'

```
value = WwGetValue(wid)
    PRINT, value
END

PRO DefaultCB, wid, data
    PRINT,'Default Item Selected'
    value = WwGetValue(wid)
    PRINT, value
END
```

Widget Commands

```
top=WwInit('ww_ex28', 'Examples', layout)
items = ['Presidents Day', 'St.Patricks Day', $
    'Easter', 'Memorial Day', '4th of July', $
    'Labor Day', 'Halloween', 'Thanksgiving', $
    'Hanukkah', 'Christmas', 'New Years Eve']
list=WwList(layout, items, 'ListCB', $
    'DefaultCB', Visible=7, /Multi)
status=WwSetValue(top, /Display)
WwLoop
```

WwLoop Function

Handles the dispatching of events and calling of PV-WAVE callbacks.

Usage

WwLoop

Parameters

None.

Returned Value

None.

Discussion

WwLoop causes PV-WAVE to loop indefinitely, processing the events and dispatching callbacks. WwLoop is always the last WAVE Widgets command in a WAVE Widgets procedure.

Example

For examples showing the use of WwLoop, see any of the WAVE Widgets widget-creation routines, such as WwButtonBox, WwCommand, WwControlsBox, WwList, and so on.

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwMainWindow Function

Creates a top-level window and a layout widget.

Usage

shell = WwMainWindow(parent, workarea, [destroyCallback])

Input Parameters

parent — The widget ID of the parent widget.

destroyCallback — A string containing the name of the callback that is executed when the main window is destroyed.

Output Parameters

workarea — The widget ID of the layout widget that is created inside of the top-level shell.

Returned Value

shell – The widget ID of the top-level shell.

Keywords

Board — If present and nonzero, a bulletin board layout is created inside the main window shell. See WwLayout for more information.

Border - Specifies the width in pixels of the borders for the layout widget and its child widgets. Default is 0.

Form — If present and nonzero, a form layout is created inside the shell. See WwLayout for more information.

Horizontal — If present and nonzero, aligns child widgets horizontally within the layout widget (the default). Used with row/ column layouts only. For more information on layout widgets, see the WwLayout function.

Position — Specifies the x, y position of the upper-left corner of the main window on the screen.

Spacing — Specifies the amount of space in pixels between child widgets inside the layout. Used with row/column layouts only. Default is 0. For more information on layout widgets, see the WwLayout function.

Title — A string specifying a title for the shell.

Vertical — If present and nonzero, aligns child widgets vertically within the layout widget. Used with row/column layouts only. For more information on layout widgets, see the WwLayout function.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font - Specifies the name of the font used for text (not supported for OLIT).

Foreground — Specifies the foreground color name.

Get/Set Value

Not supported.

Callback Parameters

Any main window callback procedure must have the following two parameters:

shell — Main window shell widget ID.

layout - Layout widget ID.

Discussion

The WwInit function creates one main window, and for many applications, this is sufficient. WwMainWindow gives you the ability to create additional main windows and layout widgets.

By default, the WwMainWindow creates a row/column layout widget. For more information on layout widgets, see WwLayout.

Example

This example shows a call to the WwMainWindow function. No callback is specified.

```
shell = WwMainWindow(parent, form, destroyCB, $
   /Vertical, Title='Topics')
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwMenuBar Function

Creates a menu bar.

Usage

menubar = WwMenuBar(parent, items)

Input Parameters

parent — The widget ID of the parent widget.

items - An unnamed structure containing the menu bar items. For more information, see Creating and Handling Menus on page 425 of the PV-WAVE Programmer's Guide.

Returned Value

menubar - The widget ID of the menu bar widget.

Input Keywords

Position — If the menu bar widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the menu bar widget within the bulletin board.

Spacing — Specifies the amount of space in pixels between child widgets.

Output Keywords

Menus — Returns an array of menu pane widget IDs in the order in which the menus were created. A menu pane is a special menu widget that serves as a container for a menu item. Menu pane widget IDs can be used in the WwMenuItem function to add, modify, or delete menu items.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the menu bar widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the menu bar widget is attached to the bottom of the parent widget.

Left - If a widget ID is specified (for example, Left=wid), then the left side of the menu bar widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the menu bar widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the menu bar widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the menu bar widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the menu bar widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the menu bar widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — Gets the button label or icon pixmap ID for the selected menu button. Or, if the button is a toggle, determines if the button is selected or unselected (selected = 1, unselected = 0).

setvalue — Sets the button label or icon pixmap ID for the selected menu button. Or, if the button is a toggle, selects or unselects the toggle button (select = 1, unselect = 0).

Callback Parameters

Any menu bar callback procedure must have the following two parameters:

```
wid - Widget ID of the selected menu item.
index — Index of the selected menu item (1-n).
```

Example

First, this example creates a menu bar with three menus: Fonts, Size, and Icons. The unnamed structure menus contains all of the information used to create the menus.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt (or enter them in a command file and run them with the @ command). To dismiss the menu bar, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

```
PRO MenuCB, wid, index
   PRINT, 'Menu Item', index, 'selected.'
   value = WwGetValue(wid)
   PRINT, value
END
```

Widget Commands

```
top=WwInit('ww_ex29', 'Examples', layout)
         menus={,callback:'MenuCB', $
            menubutton: 'Fonts', $
            menu:{,callback:'MenuCB', $
               menubutton: 'Adobe', $
               menu:{,callback:'MenuCB', $
                   toggle: 'Normal', $
                   toggle: 'Bold', $
                   toggle: 'Italic'}, $
               button: 'Helvetica', $
               button:'Courier'}, $
               menubutton: 'Size', $
               menu:{,callback:'MenuCB', $
                  button: '8', $
                  button: '10', $
                  button: '12'}, $
            menubutton: 'Icons', $
            menu:{,callback:'MenuCB', $
               title: 'Help', $
icon:getenv('WAVE DIR')+ $
   '/src/help/app/xres/wxbm btn help search.', $
icon:getenv('WAVE DIR')+ $
   '/src/help/app/xres/wxbm btn help toc.', $
icon:getenv('WAVE DIR')+ $
   '/src/help/app/xres/wxbm btn help_topics.', $
separator:1,
icon:getenv('WAVE DIR')+ $
   '/src/help/app/xres/wxbm_btn_help_quit.'}}
         bar=WwMenuBar(layout, menus)
         status=WwSetValue(top, /Display)
         WwLoop
```

See Also

WwMenuItem, WwOptionMenu, WwPopupMenu

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwMenuItem Function

Adds, modifies, or deletes specified menu items.

Usage

status = WwMenuItem(parent, item, value [, callback])

Input Keywords

parent — The menu pane widget ID acquired using the Menus keyword from the WwMenuBar, WwOptionMenu, or WwPopupMenu functions. A menu pane is a special menu widget that serves as a container for a menu item.

item - The index of the menu item; the index of the first menu item is one (1). Used for an Update or Delete operation.

value — The value (string) of the menu item. Used for an Update or Add operation.

callback — The PV-WAVE callback procedure that is executed when the menu item is selected. Used for an Add operation.

Returned Value

status — Returns one (1) if the function is successful, or zero (0) if the function is not successful.

Keywords

Update – Modifies the value of the item specified by the index of item.

Add — Appends the specified item. The type of the item is specified by additional keywords:

- Button A push button item is added.
- Toggle A toggle (radio) button is added.
- *Icon* A graphic (icon) button is added.

Delete – Deletes the item specified by *index*.

Discussion

WwMenuItem lets you dynamically update menus that have already been created. All menu items are placed inside a parent menu pane, and the widget ID of the appropriate menu pane can be acquired using the *Menus* keyword of the WwMenuBar, WwPopupMenu, or WwOptionMenu functions.

To update (*Update* keyword) or remove (*Delete* keyword) a menu item, use the appropriate menu item index; the index of the first menu item is one (1). To add a menu item to the bottom of the menu use the Add keyword.

Example

The following fragment shows how a callback might be written that uses WwMenuItem to modify the contents of a menu pane.

```
PRO ButtonCB, wid, index
   COMMON Menus, menupane, item no, new name
   CASE index OF
   1: BEGIN ; Add new item
   status = WwMenuItem(menupane, item no,$
      new name, /Add)
      END
   2: BEGIN: ; Update last selected item
   status = WwMenuItem(menupane, item no, $
      new name, /Update)
     END
```

```
3: BEGIN ; Remove last selected item
   status = WwMenuItem(menupane, item no, $
      /Delete)
      END
   ENDCASE
END
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwMessage Function

Creates a blocking or nonblocking message box.

Usage

wid = WwMessage(parent, label, OKCallback, CancelCallback, HelpCallback)

Input Parameters

```
parent — The widget ID of the parent widget.
```

label – A string containing the message text.

OKCallback - A string containing the name of the callback routine that is called when the OK (Motif) or Confirm (OPEN LOOK) button is selected.

CancelCallback - A string containing the name of the callback routine that is executed when the Cancel button is selected.

HelpCallback — A string containing the name of the callback that is executed when the Help button is selected. If you are running under OPEN LOOK, this parameter is ignored.

Keywords

Block — If present and nonzero, creates a blocking message box (the default).

Info — If present and nonzero, creates an "information" message box (the default, Motif only).

Nonblock — If present and nonzero, creates a nonblocking message box.

Question — If present and nonzero, creates a "question" message box (Motif only).

Title - Specifies a string containing the message box title.

Warning — If present and nonzero, creates a "warning" message box (Motif only).

Working — If present and nonzero, creates a "working" message box (Motif only).

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font - Specifies the name of the font used for message text.

Foreground — Specifies the foreground color name.

Returned Value

wid — The message box widget ID.

Get/Set Value

Not supported.

Callback Parameters

Any message box callback procedure must have the following two parameters:

```
wid - Message widget ID.
shell — Shell widget ID.
```

Discussion

A message window is a *popup* window. This means that it cannot be the child of the top-level shell or the layout widget. Usually, a message widget is activated by a pushbutton or menu button, as in the example below.

Example

This example creates a button box with four buttons. If you are running under Motif, each button activates one of the four types of message windows: information, working, warning, or question. (Under OPEN LOOK, the four message boxes will look the same.) The callback MessageOK is executed when the user clicks on the OK or Confirm button. The callback MessageCancel is executed when the user clicks on the Cancel button.

Enter the callback procedures into a file, and compile them with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the widgets, select the appropriate function (such as Close) from the window manager menu of the menu bar.

Callback Procedures

```
PRO MbuttonCB, wid, data
CASE data OF
   1: message=WwMessage(wid, $
      'This is a Test Message', 'MessageOK', $
      'MessageCancel', Title='Information')
   2: message=WwMessage(wid, $
      'This is a Test Message', 'MessageOK', $
```

```
'MessageCancel', /Working, $
      Title='Working')
   3: message=WwMessage(wid, $
      'This is a Test Message', 'MessageOK', $
      'MessageCancel', /Warning, $
      Title='Warning')
   4: message=WwMessage(wid, $
      'This is a Test Message', 'MessageOK', $
      'MessageCancel', /Question, $
      Title='Question')
 ENDCASE
END
PRO MessageOK, wid, data
   PRINT, 'Message OK'
END
PRO MessageCancel, wid, data
   PRINT, 'Message Cancel'
END
```

Widget Commands

```
top=WwInit('ww_ex30', 'Examples', layout)
button=WwButtonBox(layout, ['Information', $
    'Working', 'Warning', 'Question'], $
    'MbuttonCB')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwOptionMenu Function

Creates an option menu.

Usage

option = WwOptionMenu(parent, label, items)

Input Parameters

parent — The widget ID of the parent widget.

label — A string containing the text of the option menu label.

items — An unnamed structure specifying the option menu items. For more information, see Creating and Handling Menus on page 425 of the PV-WAVE Programmer's Guide.

Returned Value

option — The widget ID of the option menu box.

Input Keywords

Position — If the option menu widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the option menu widget within the bulletin board.

Output Keywords

Menus — Returns an array of menu pane widget IDs in the order in which the menus were created. A menu pane is a special menu widget that serves as a container for a menu item. Menu pane widget IDs can be used in the WwMenuItem function to add, modify, or delete menu items.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font - Specifies the name of the font used for label and button text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the option menu widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the option menu widget is attached to the bottom of the parent widget.

Left – If a widget ID is specified (for example, Left=wid), then the left side of the option menu widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the option menu widget is attached to the left side of the parent widget.

Right - If a widget ID is specified (for example, Right=wid),then the right side of the option menu widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the option menu widget is attached to the right side of the parent widget.

Top - If a widget ID is specified (for example, Top=wid), then the top of the option menu widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the option menu widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — The button label or icon pixmap ID for the selected menu button. Or, if the button is a toggle, returns whether it is selected or unselected (selected = 1, unselected = 0).

setvalue — The button label or icon pixmap ID for the selected menu button. Or, if the button is a toggle, selects or unselects the toggle button (select = 1, unselect = 0).

Caliback Parameters

Any option menu callback procedure must have the following two parameters:

wid — Widget ID of the selected menu item.

index — Index of the selected menu item (1-n).

Discussion

An option menu is a button that, when selected, displays a menu. The current selection is always displayed on the option menu button. When the user makes a selection, the option menu button is updated to reflect the change.

Pullright menus are not allowed in an option menu.

Example

This example creates an option menu. The menu information is defined in an unnamed structure called fonts. The callback routine, MenuCB, is called whenever a menu item is selected.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the option menu, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

```
PRO MenuCB, wid, index
    PRINT, 'Menu Item', index, 'selected.'
    value = WwGetValue(wid)
    PRINT, value
END
```

Widget Commands

```
top=WwInit('ww_ex31', 'Examples', layout)
fonts={,callback:'MenuCB', $
    button:'Adobe',$
    button:'Helvetica',$
    button:'Courier'}

opmenu=WwOptionMenu(layout, 'Fonts:', $
    fonts, Position=[0,150])
status=WwSetValue(top, /Display)
WwLoop
```

See Also

WwMenuItem, WwMenuBar, WwPopupMenu

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwPopupMenu Function

Creates a popup menu.

Usage

popup = WwPopupMenu(parent, items)

Input Parameters

parent - The widget ID of the parent widget.

items — An unnamed structure specifying the menu bar items. For more details, see Creating and Handling Menus on page 425 of the PV-WAVE Programmer's Guide.

Returned Value

popup — The widget ID of the popup menu widget.

Input Keywords

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Output Keywords

Menus - Returns an array of menu pane widget IDs in the order in which the menus were created. A menu pane is a special menu widget that serves as a container for a menu item. Menu pane widget IDs can be used in the WwMenuItem function to add, modify, or delete menu items.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue — The button label or icon pixmap ID for the selected menu button. Or, if the button is a toggle, returns whether it is selected or unselected (selected = 1, unselected = 0).

setvalue — The button label or icon pixmap ID for the selected menu button. Or, if the button is a toggle, selects or unselects the toggle button (select = 1, unselect = 0).

Callback Parameters

Any popup menu callback procedure must have the following two parameters:

wid — Widget ID of the selected menu item.

index – Index of the selected menu item (1-n).

Discussion

A popup menu is a menu that appears when the user presses a mouse button (usually the right button) when the pointer is in the popup menu's parent widget. Thus, the popup menu is not tied to any menu button or menu bar.



Avoid creating popup menus that have a multi-line text widget as a parent. The system editing menu may appear over the popup menu that you have defined.

Example

This example creates a drawing widget that serves as the parent for a popup menu. The popup menu is activated when the right mouse button is pressed while the pointer is over the drawing widget. The callback procedure is executed when a menu button is selected.

Enter the callback procedures into a file, and compile the procedures with the .RUN command. Then, enter the widget commands at the WAVE> prompt (or enter them in a command file and run them with the @ command). To dismiss the button box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedures

```
PRO DrawCB, wid, data
   COMMON draw, img
   PRINT, 'Draw'
   TV, img
END
PRO MenuCB, wid, index
   PRINT, 'Menu Item', index, 'selected.'
   VALUE = WwGetValue(wid)
   PRINT, value
END
```

Widget Commands

```
top=WwInit('ww ex32', 'Examples', layout)
COMMON draw, img
LOADCT, 5, /SILENT
img=BYTARR(512,512)
OPENR, 1, '$WAVE DIR/data/head.img'
   Note: Under VMS, enter: WAVE DIR:[DATA]head.img
READU, 1, img
CLOSE, 1
draw=WwDrawing(layout, 1,'DrawCB', $
   [256,256], [512,512])
status=WwSetValue(top, /Display)
menus={,callback:'MenuCB', $
   menubutton: 'Fonts', $
   menu:{,callback:'MenuCB', $
```

```
menubutton:'Adobe', $
menu:{,callback:'MenuCB', $
    toggle:'Normal', $
    toggle:'Bold', $
    toggle:'Italic'}, $
   button:'Helvetica', $
   button:'Courier'}, $
   menubutton:'Size', $
   menu:{,callback:'MenuCB', $
    button:'8', $
   button:'10', $
   button:'12'}}

menu=WwPopupMenu(draw, menus)
status=WwSetValue(top, /Display)
WwLoop
```

See Also

WwMenuItem, WwMenuBar, WwOptionMenu

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwRadioBox Function

Creates a box containing radio buttons.

Usage

radio = WwRadioBox(parent, labels, callback)

Input Parameters

```
parent — The widget ID of the parent widget.labels — A string array of button labels.
```

callback — A string containing the name of the PV-WAVE callback that is executed when a radio button is selected or unselected.

Returned Value

radio — The widget ID of the radio box widget. If only one button is requested, that radio button's widget ID is returned.

Input Keywords

Border — Specifies the width in pixels of the radio box and button borders.

Horizontal — When present and nonzero, creates a horizontally aligned row of radio buttons.

Measure — Specifies the number of columns of radio buttons (for a vertical box) or rows (for a horizontal box).

Nofmany — When present and nonzero, creates nonexclusive radio buttons, where any number of buttons can be selected at once.

Oneofmany — When present and nonzero, creates exclusive radio buttons, where only one button can be selected at a time.

Position — If the radio box widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the radio box widget within the bulletin board.

Spacing — Specifies the space in pixels between radio buttons.

Vertical — When present and nonzero, creates a vertically aligned column of radio buttons.

Output Keywords

Toggles — Returns an array of toggle button widget IDs.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor – Specifies the base color.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the radio box widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the radio box widget is attached to the bottom of the parent widget.

Left – If a widget ID is specified (for example, Left=wid), then the left side of the radio box widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the radio box widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the radio box widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the radio box widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the radio box widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the radio box widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

```
getvalue — Gets the state of the selected radio button. Set = 1;
unset = 0.
```

setvalue — Sets the state of the selected radio button. Set = 1; unset = 0.

Callback Parameters

Any radio button callback procedure must have the following two parameters:

```
wid - Toggle button widget ID.
```

index — Index of the toggle button changed (1 - n).

Discussion

This function creates a box containing a specified number of rows or columns of labeled toggle buttons. If only one button is created, a box is not created; instead, only a single button widget is created.

Example

This example creates a box containing radio buttons. The callback is executed whenever the user clicks on a button.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt. To dismiss the radio button box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

```
PRO RadioCB, wid, which
 CASE which OF
   1: PRINT, 'First Toggle Selected'
   2: PRINT, 'Second Toggle Selected'
```

```
3: PRINT, 'Third Toggle Selected'
ENDCASE

value = WwGetValue(wid)

PRINT, value
END
```

Widget Commands

```
top=WwInit('ww_ex33', 'Examples', layout)
labels=['System','Owner','Group']
rbox=WwRadioBox(layout,labels, 'RadioCB', $
    /Vertical, Border=2, Spacing=20)
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwSetValue Function

Sets the specified value for a given widget.

Usage

```
status = WtSetValue(widget, [value])
```

Input Parameters

```
widget — The ID of the widget whose value you want to set.
value — The value of the widget (optional when keywords are specified).
```

Returned Value

status — Returns 1 if the function is successful, or 0 if the function is not successful.

Keywords

Close – If present and nonzero, closes the widget hierarchy from the top-level shell down.

Display - If present and nonzero, displays the widget hierarchy from the top-level shell down.

Hide - If present and nonzero, hides the specified widget or group of widgets (layout).

Nonsensitive - If present and nonzero, sets the widget to nonsensitive.

Position — If the widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the widget within the bulletin board.

Show - If present and nonzero, shows the specified widget or group of widgets (layout).

Sensitive — If present and nonzero, sets the widget to sensitive.

Userdata — Stores the specified PV-WAVE variable with the widget. You can retrieve this value later with WwGetValue.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor – Specifies the base color.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom=wid), then the bottom of the widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the widget is attached to the top of the parent widget.

Discussion

See the *Get/Set Value* section under each WAVE Widget function description to find out what value is set by WwSetValue for each function. For example, WwSetValue called with the ID of a list widget sets the current list of items to a specified string array of new items.

Example

The following example demonstrates two common uses of WwSetValue: displaying and closing widgets. In the callback rou-

tine CommandDone, WwSetValue is called to close the widget when the user selects the Close function from the window manager menu. In the Widget Commands section, WwSetValue is used to display the button box widget.

```
PRO CbuttonCB, wid, data
   command = WwCommand(wid, 'CommandOK', $
      'CommandDone', Position=[300,300], $
      Title='Command Entry Window')
END
PRO CommandOK, wid, shell
   value = WwGetValue(wid)
   PRINT, value
END
PRO CommandDone, wid, shell
   status = WwSetValue(shell, /Close)
END
```

Widget Commands

```
top=WwInit('ww_ex34', 'Examples', layout)
button=WwButtonBox(layout, 'Command', $
   'CbuttonCB')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwTable Function

Creates an editable 2D array of cells containing string data, similar to a spreadsheet.

Usage

table = WwTable(parent, callback [, variable])

Input Parameters

parent - The widget ID of the parent widget.

callback — A PV-WAVE procedure that is called when the contents of a cell are modified.



It is the calling routine's responsibility to modify the appropriate PV-WAVE variables when a table cell has been modified.

variable — (optional) A PV-WAVE variable used to initialize the table cells. When this parameter is not supplied, the cells are initially empty, and the size of the table is set to the size specified by the Cols and Rows keywords. You can use the following types of variables with WwTable:

- scalar
- vector
- 2D array
- structure with scalar fields or date/time structure field

All values are converted to type string within WwTable.

Returned Value

table — The widget ID of the table widget.

Keywords

Alignments — A one-dimensional array (0, ..., cols-1) of column alignments. Valid values are:

- 0 Align cell contents to cell's left edge (left justify).
- 1 Center cell contents (center justify).
- 2 Align cell contents to cell's right edge (right justify).

Clabels – A one-dimensional string array (0, ..., cols-1) of column labels.



It is recommended that you specify row/column labels. Clicking MB2 on a row label selects the whole row; clicking MB2 on a column label selects the whole column.

Colors — A two-dimensional array (0, ..., rows-1, 0, ..., cols-1) of color indexes for table cells. This enables you to highlight particular groups of cells.

Cols – The number of columns in the table. If the Cols keyword is not specified, and the variable parameter is specified, the number of columns is calculated from the dimensions of variable. If neither Cols nor variable are specified, the size of the table is set to one column.

Cwidth - A one-dimensional array (0, ..., cols-1) of column widths. If not specified, the default column width is 10 characters.

Fixrows — The number of fixed rows in the table. Fixed rows are non-scrollable, non-editable cells that can serve as labels.

Fixeds — The number of fixed columns in the table. Fixed columns are non-scrollable, non-editable cells that can serve as labels.

Note //

You can specify Fixrows or Fixcols for a table, but not both.

Horizontal — If this keyword is present and nonzero, the contents of *variable* are displayed in natural fashion, i.e., *variable* rows are horizontal and columns are vertical. *Horizontal* is enabled by default. The *Horizontal* and *Vertical* keywords are mutually exclusive.

Position — If the widget is to be placed in a bulletin board layout, use this keyword (a two-element vector) to specify the x, y coordinates of the widget within the bulletin board.

Rlabels — A one-dimensional string array (0, ..., rows-1) of row labels.

Rows — The number of rows in the table. If the *Rows* keyword is not specified, and the *variable* parameter is specified, the number of rows is calculated from the dimensions of *variable*. If neither *Rows* nor *variable* are specified, the size of the table is set to one row.

Setcelldata — A copy of any PV-WAVE variable to be passed as client data to the Setcells function.

Setcells — Allows you to name a PV-WAVE function that is responsible for setting the values of the exposed cells. This function is called, like a callback, whenever cells are exposed (for example, as the user scrolls through the table). The function returns a string that is used to set the cell's new value. The function's input parameters are the table widget ID, a vector [row, column] specifying the exposed cell, and client data specified with the Setcelldata keyword. If this function is specified, the variable parameter is ignored.

Using Setcells can improve performance when a large table is created. Ordinarily, when the table's cells are populated by values taken from a single variable (the variable parameter), WwTable actually copies this variable first. When Setcells is used, only the values currently displayed are held in memory. See Example 2 on page 479 for more information on Setcells.

Vertical — If this keyword is present and nonzero, the contents of variable are displayed as transposed (variable rows are vertical and columns are horizontal). The Horizontal and Vertical keywords are mutually exclusive.

Visible — A two-element vector specifying the number of rows and columns displayed. If the table size is bigger than the number of visible rows and columns, scrollbars are placed at the right and bottom edges of the window, so that you can view different portions of the table. If this keyword is not specified, four rows and four columns are displayed.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font - Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the widget is attached to the top of the parent widget.

Get/Set Value

getvalue — Gets the Boolean array (0...rows-1, 0...cols-1) of selected cells. For information on cell selection, see the Discussion section.

setvalue — A two-element vector [row, column] the indices of which select the cell to be set.

Callback Parameters

Any table widget callback procedure must have the following three parameters:

wid — The table widget ID.

cell — A two-element vector [*row*, *column*] specifying the modified cell.

value – A string containing the new contents of the modified cell.

Discussion

This section discusses how to select table cells for editing and for other operations.

Selecting Cells for Editing

When a cell is selected for editing, it is considered the current cell. You can delete and add characters in the current cell using the keys on your keyboard.

- To edit a cell, select it by clicking the left mouse button in the
- To edit the cell to the left of the current cell, press the <Shift>+<Tab> key.
- To edit the cell to the right of the current cell, press the <Tab>
- To edit the cell above the current cell, press the up arrow key.
- To edit the cell on the bottom of the current cell, press the down arrow key.

Selecting One or More Cells

This section describes how to select cells for operations other than editing.

- To select a single cell, click the middle mouse button on the cell.
- To extend the selection, press the <Shift> key and click the middle mouse button.
- To deselect the cell, press the <Shift>+<Control> keys and click the middle mouse button.
- To toggle the selection press the <Control> key and click the middle mouse button.
- To select a rectangular region of cells, press the middle mouse button, drag the mouse, and then release the middle mouse button.

 To select, deselect, extend, or toggle the selection of an entire row or column, click the middle mouse button (with appropriate key(s) listed above pressed) on the row or column label.

Example 1

The following example displays data stored in the phone_data variable in the save file:

```
$WAVE_DIR/data/phone_example.sav
To restore this file, enter the following commands:
    .RUN
    RESTORE, !Dir+'/data/phone_example.sav'
    COMMON Tablecomm, phone_data
    END
```

Callback Procedures

```
PRO tableCB, wid, which, text
COMMON Tablecomm, phone data
   PRINT, 'Table', which, text
   status = WwSetValue(wid, which)
   PRINT, WwGetValue(wid)
END
PRO ButtonCB, wid, data
COMMON Tablecomm, phone data
   PRINT, 'Table Selected'
   shell = WwMainWindow(wid, form, /Vertical, $
      TITLE = 'Table')
   clrs = INTARR(N TAGS(phone data), $
      N ELEMENTS (phone data))
   FOR i = 0, N ELEMENTS(phone data)-1 DO $
      clrs(*, i) = INDGEN(N TAGS(phone data)) * 20
  rlabs = $
  STRTRIM(STRING(INDGEN(N ELEMENTS(phone data))), 2)
```

```
table = WwTable(form, 'tableCLBK', phone_data, $
      Colors=clrs, Visible=[10, N_TAGS(phone_data)-4], $
      Clabels=TAG NAMES(phone data), Rlabels = rlabs, $
      /Vertical)
   status = WwSetValue(shell, /Display)
END
```

Widget Commands

```
top=WwInit('tble ex', 'Examples', layout)
label = ['Table']
button=WwButtonBox(layout, label, 'ButtonCB')
status=WwSetValue(top, /Display)
WwLoop
```

Example 2

This example demonstrates the use of the Setcells and Setcelldata keywords. In this program fragment, the contents of table cells are set to the value of the row number times the column number.

Callback Procedures

```
FUNCTION settableCLBK, wid, data, n, r, c
   RETURN, STRTRIM(STRING(r * c), 2)
      Return the string value of the product of row times column.
END
PRO buttonCB, wid, data
  COMMON Tablecomm, phone_data
   PRINT, 'Table Selected'
   shell = WwMainWindow(wid, form, /VERTICAL, $
      TITLE = 'Table')
```

```
table = WwTable(form, 'tableCLBK', Visible=[10,5], $
Rows=40, Cols=50, Setcells = 'settableCLBK', $
Setcelldata='any var')
status = WwSetValue(shell, /Display)
END
```

Widget Commands

```
top=WwInit('tble_ex20', 'Examples', layout)
label = ['Table']
button=WwButtonBox(layout, label, 'buttonCB')
status=WwSetValue(top, /Display)
WwLoop
```

See Also

WtTable

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, *Using WAVE Widgets*, in the *PV-WAVE Programmer's Guide*.

WwText Function

Creates a text widget that can be used for both single-line text entry or as a full text editor. In addition, this function can create a static text label.

Usage

text = WwText(parent, verifyCallback)

Input Parameters

parent — The widget ID of the parent widget.

verifyCallback - A string containing the name of a callback routine that is executed when the contents of the text field changes.

Keywords

Cols - Specifies the number of columns in the text field (in characters).

File - Specifies a name of a file containing text.

Label — If present and nonzero, the widget is a label (static text). The value of the label is set using the *Text* keyword. If the *Label* keyword is set to a string value, the text field (single line) or text edit (multi-line) widget is preceded by this string.

Pixmap — Specifies a filename containing a pixmap or bitmap file to be displayed as a label. This keyword is used in conjunction with the *Label* keyword.

Position — If the text widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the text widget within the bulletin board.

Read — If present and nonzero, the text widget is read only.



Because OLIT does not support read-only text fields, the staticText OLIT widget is used for the OLIT version of the readonly text field widget.

Rows - Specifies the number of rows in the text field.

Text – Specifies a label for the text field.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Font – Specifies the name of the font used for text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom=wid), then the bottom of the text widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the text widget is attached to the bottom of the parent widget.

Left — If a widget ID is specified (for example, Left=wid), then the left side of the text widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the text widget is attached to the left side of the parent widget.

Right — If a widget ID is specified (for example, Right=wid), then the right side of the text widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the text widget is attached to the right side of the parent widget.

Top — If a widget ID is specified (for example, Top=wid), then the top of the text widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the text widget is attached to the top of the parent widget.

Returned Value

text — The text widget ID.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

getvalue - A string containing the displayed text.

setvalue — A string containing the displayed text.

Callback Parameters

Any text widget callback procedure must have the following two parameters:

wid - Text widget ID.

parent — Parent widget ID.

Example 1: Single-line Text Field and Label

This example creates a layout widget containing a single-line text field and a label. The callback is executed when the user enters text in the text field and presses <Return>.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt (or enter them in a command file and run it with the @ command). To dismiss the button box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

PRO TextCB, wid, data

```
PRINT, 'Text done'
value = WwGetValue(wid)
PRINT, value
END
```

Widget Commands

```
top=WwInit('ww_ex35', 'Examples', layout, $
    /Form)
    Initialize WAVE Widgets and create the form layout widget.
label=WwText(layout, /Label, $
    Text='This is Label')
        Create the label widget.
text=WwText(layout, 'TextCB', Cols=40, $
    left=label)
        Create the single-line text field widget, attaching it to the right edge of the label widget.
status=WwSetValue(top, /Display)
WwLoop
```

Example 2: Multi-line Text Window

This example creates a multi-line text window. Because the *Read* keyword is used, the text is read-only.

Callback Procedure

```
PRO TextCB, wid, data
    PRINT, 'Text done'
    value = WwGetValue(wid)
    PRINT, value
END
```

Widget Commands

```
top=WwInit('ww_ex36', 'Examples', layout)
filename = getenv('WAVE_DIR')+ $
   '/Tips'
```

```
text=WwText(layout, 'TextCB', /Read, $
   File=filename, Cols=40, Rows=20)
status=WwSetValue(top, /Display)
WwLoop
```

See Also

WgTextTool

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

WwToolBox Function

Creates an array of graphic buttons (icons).

Usage

toolb = WwToolBox(parent, labels, callback)

Input Parameters

parent - The widget ID of the parent widget.

labels — A string array of icon file names.

callback - A string containing the name of the PV-WAVE callback routine that is executed when a button is selected.

Returned Value

toolb — The ID of the tool box widget. If only one button is requested, that button's widget ID is returned.

Input Keywords

Border — Specifies the width in pixels of the tool box and icon borders. The default is 0.

Horizontal – When present and nonzero, creates a horizontally aligned row of buttons (the default).

Measure - If you create a horizontal box, then measure specifies the number of rows into which the buttons will be divided. For a vertical box, this keyword specifies the number of columns in which to divide the buttons.

Position — If the tool box widget is to be placed in a bulletin board layout, use this keyword to specify the x, y coordinates of the tool box widget within the bulletin board.

Spacing – Specifies the space in pixels between buttons. The default is 0.

Vertical — When present and nonzero, creates a vertically aligned column of buttons.

Output Keywords

Tools – Returns an array of graphical button (icon) widget IDs.

Color/Font Keywords

For additional information on the color and font keywords, see Setting Colors and Fonts on page 463 of the PV-WAVE Programmer's Guide.

Background — Specifies the background color name.

Basecolor — Specifies the base color.

Font – Specifies the name of the font used for button text.

Foreground — Specifies the foreground color name.

Attachment Keywords

For additional information on attachment keywords, see Form Layout: Attachments on page 422 of the PV-WAVE Programmer's Guide.

Bottom — If a widget ID is specified (for example, Bottom= wid), then the bottom of the tool box widget is attached to the top of the specified widget. If no widget ID is specified (for example, /Bottom), then the bottom of the tool box widget is attached to the bottom of the parent widget.

Left - If a widget ID is specified (for example, Left=wid), then the left side of the tool box widget is attached to the right side of the specified widget. If no widget ID is specified (for example, /Left), then the left side of the tool box widget is attached to the left side of the parent widget.

Right - If a widget ID is specified (for example, Right=wid),then the right side of the tool box widget is attached to the left side of the specified widget. If no widget ID is specified (for example, /Right), then the right side of the tool box widget is attached to the right side of the parent widget.

Top - If a widget ID is specified (for example, Top=wid), then the top of the tool box widget is attached to the bottom of the specified widget. If no widget ID is specified (for example, /Top), then the top of the tool box widget is attached to the top of the parent widget.

Get/Set Value

For information on Get and Set values, see Setting and Getting Widget Values on page 466 of the PV-WAVE Programmer's Guide.

```
getvalue – Icon (pixmap) ID.
setvalue — The icon file name.
```

Callback Parameters

Any toolbox callback procedure must have the following two parameters:

```
wid - Button widget ID.
index — Index of the button pushed (1-n).
```

Discussion

This function creates a box containing iconic buttons arranged in rows and/or columns. If you only create a single button, then the box widget is not created — only the button widget is created.

Example

This example creates a box containing graphical buttons (a toolbox). The callback routine, PickedCB, is executed when one of the toolbox buttons is selected.

Enter the callback procedure into a file, and compile the procedure with the .RUN command. Then, enter the widget commands at the WAVE> prompt (or enter them in a command file and run them with the @ command. To dismiss the tool box, select the appropriate function (such as Close) from the window manager menu.

Callback Procedure

```
PRO PickedCB, wid, which

CASE which OF

1: PRINT, 'Search Selected'

2: PRINT, 'Toc Selected'

3: PRINT, 'Topics Selected'

4: PRINT, 'Quit Selected'

ENDCASE

END
```

Widget Commands

```
top=WwInit('ww_ex37', 'Examples', layout)
pixmaps = [GETENV('WAVE DIR')+ $
   'src/help/app/xres/wxbm_btn_help_search.',$
    getenv('WAVE DIR')+ $
   'src/help/app/xres/wxbm_btn_help_toc.',$
    qetenv('WAVE DIR')+ $
   'src/help/app/xres/wxbm btn_help_topics.',$
    getenv('WAVE DIR')+ $
   'src/help/app/xres/wxbm_btn_help_quit.']
dbox=WwToolBox(layout, pixmaps, 'PickedCB', /Vertical, $
   Spacing=20, Measure=2)
status=WwSetValue(top, /Display)
WwLoop
```

See Also

For more information about how to write an application program based on WAVE Widgets, refer to Chapter 15, Using WAVE Widgets, in the PV-WAVE Programmer's Guide.

XYOUTS Procedure

Draws text on the currently selected graphics device starting at the designated data coordinate.

Usage

XYOUTS, x, y, string

Input Parameters

x, y - x is the column, and y is the row, at which the output string should start. x and y are normally taken to be in data coordinates; however, the *Device* and *Normal* keywords can be used to change this unit.

Output Parameters

string — The scalar string containing the text that is to be output to the display surface. If not of string type, it is converted prior to use.

Keywords

XYOUTS keywords are listed below. For a description of each keyword, see Chapter 3, *Graphics and Plotting Keywords*.

Alignment Channel	Color	Noclip	Text_Axes
Charrize	Data	Normal	Thick
	Device	Orientation	Width
Charthick	Font	Size	Z
Clip	Linestyle	T3d	

Discussion

XYOUTS is machine-dependent when you are using hardware fonts. This means that on two different machines, the same PV-WAVE commands may produce text that does not appear the same. To guarantee similar appearance, use software fonts.



You may notice that under X Windows the size of the software fonts varies from device to device. When you start PV-WAVE, the PV-WAVE hardware font is set to the current hardware font of the X server. Not all X servers will have the same default font size because users can reconfigure the default font and the default font can differ between X servers. Therefore, you may discover that the hardware font size, and therefore the software font size, may vary across different workstations. You can avoid this by explicitly setting the X font using the DEVICE procedure. For example:

```
DEVICE, font='-adobe-courier-medium-r-normal-
   -14-*'
```

Example

In this example, XYOUTS is used to label a plot of random data. Procedure XYOUTS also is used to place a title on the plot. Text placement is relative to the x and y coordinates of the plot, which is the default for XYOUTS. Note that the CURSOR procedure was used to determine the proper coordinates at which to place labels. This example uses PV-WAVE Advantage procedure RAN-DOMOPT.

```
RANDOMOPT, Set = 1234567
x = RANDOM(5)
   Create a 5-element vector of random data.
PLOT, x, Xrange = [-0.5, 4.5]
   Plot the data.
XYOUTS, 0.1, 0.65, "Point 1"
   Label the first data point.
XYOUTS, 1.1, 0.20, "Point 2"
   Label the second data point.
```

XYOUTS, 2.0, 0.425, "Point 3" Label the third data point.

XYOUTS, 3.1, 0.30, "Point 4" Label the fourth data point.

XYOUTS, 4.1, 0.825, "Point 5" Label the fifth data point.

XYOUTS, 1.25, 0.9, "Random Data", Charsize = 2 Place a title on the plot. Make the title twice the default character size using the Charsize keyword.

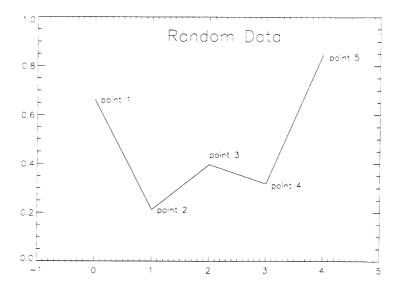


Figure 2-83 Example of plot labeling using XYOUTS.

See Also

LEGEND, PLOT

For more information, see *Using XYOUTS to Annotate Plots* on page 69 of the *PV*-WAVE User's Guide.

ZOOM Procedure

Expands and displays part of an image (or graphic plot) from the current window in a second window.

Usage

ZOOM

Parameters

None.

Input Keywords

Fact – The zoom expansion factor. The default is 4.

Interp - Specifies the interpolation method to be used. If nonzero, uses the bilinear interpolation method. Otherwise, uses the nearest neighbor method (the default).

XSize — The X size of the new window. The default is 512.

YSize - The Y size of the new window. The default is 512.

Continuous – If set to 1, causes the zoom window to track the mouse cursor. This obviates the need to press the left mouse button to mark the center of the zoom. Note that Continuous works well only on fast computers.

Discussion

ZOOM works only on windowing systems. It provides a quick way to get a close look at your image.

ZOOM lets you click with your mouse button on the center point of a region in a previously displayed image to bring up an enlarged view of this region in a new window. Then use your mouse buttons as follows:

Use the left mouse button to choose the center point of the region to be zoomed in on from the original image.

- Use the middle mouse button to display a window letting you choose the zoom factor to use.
- Use the right button to quit out of ZOOM.

Example

```
OPENR, 1, !Data_dir + 'mandril.img'
mandril = BYTARR(512, 512)
READU, 1, mandril
Read in the PV=WAVE mandril demo image file.
```

TVSCL, mandril
Display the image file.

ZOOM

Use ZOOM with the default values; the mouse button functions will be described in the original window.

ZOOM, Fact=2
Use ZOOM with the zoom factor set to 2.

ZOOM, Interp=0

Use ZOOM with the zoomed region displayed using nearest neighbor sampling.

ZOOM, Interp=1

Use ZOOM with the zoomed region displayed using bilinear interpolation.

ZOOM, XSize=200, YSize=200

Display the new image in a window that is 200-by-200 pixels.

ZOOM, /Continuous

Run ZOOM where it continually samples the cursor; there is no need to click.

See Also

TVRD, ROT, ROT_INT

For details on interpolation methods. see *Efficiency and Accuracy of Interpolation* on page 170 of the *PV*-WAVE User's Guide.

ZROOTS Procedure

Finds the roots of the m-degree complex polynomial, using Laguerre's method.

Usage

ZROOTS, a, roots [, polish]

Input Parameters

a - A vector containing the m + 1 coefficients of the polynomial. May be either real or complex.

polish — Specifies whether polishing is to be done. Set to 0 if you want to prevent polishing of the roots. If set to 1 or omitted, roots are polished.

Output Parameters

roots — The result of ZROOTS, which is set to an m-element complex vector on exit.

Keywords

None.

Discussion

ZROOTS returns the roots of the *m*-degree complex polynomial:

$$\sum_{i=0}^{m} a_i x^i$$

See Also

POLY

ZROOTS is based on a routine of the same name in *Numerical Recipes in C: The Art of Scientific Computing*, by Flannery, Press, Teukolsky, and Vetterling, Cambridge University Press, Cambridge, MA, 1988. It is used by permission.

CHAPTER

Graphics and Plotting Keywords

This chapter describes the keywords that can be used with the PV-WAVE graphics and plotting system routines. For information on the corresponding system variables that are listed for some keywords, see Chapter 4, *System Variables*.

Alignment Keyword

Used With Routines: XYOUTS

Corresponding System Variable: None.

Specifies the horizontal alignment of the text in relation to the point x, y, which is specified as input to the XYOUTS procedure.

An alignment of 0.0 (the default) places the left edge of the text on the given (X, Y) coordinate (left-justifies). An alignment of 1.0 right-justifies the text, while 0.5 centers the text over point (X, Y).

Ax Keyword

Used With Routines: SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: None.

Specifies the angle of rotation about the X axis, in degrees, towards the viewer.

The Ax keyword parameter defaults to +30 degrees if omitted and !P.T3d is 0.



This keyword is effective only if !P.T3d is *not* set. If !P.T3d is set, the three-dimensional to two-dimensional transformation used by SURFACE is contained in the 4-by-4 array !P.T.

The surface represented by the two-dimensional array is first rotated, Az (see the next section) degrees about the Z axis, then by Ax degrees about the X axis, tilting the surface towards the viewer (Ax > 0), or away from the viewer.

The three-dimensional to two-dimensional transformation represented by Ax and Az can be saved in !P.T by including the Save plotting keyword.

Az Keyword

Used With Routines: SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: None.

Specifies the counterclockwise angle in degrees of rotation about the Z axis (when looking down the Z axis toward the origin).

This keyword is effective only if !P.T3d is *not* set. The order of rotation is Az first, then Ax.

Background Keyword

Used With Routines: CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !P.Background.

The background color index to which the screen is set when the ERASE procedure is called.



Not all devices support erasing the background to a color index.

Example

To produce a black plot with a white background on a color display:

PLOT, y, Background = 255, Color = 0

Bottom Keyword

Used With Routines: SURFACE

Corresponding System Variable: None.

The color index used to draw the lower part of the surface. If not specified, the bottom is drawn with the same color as the top.

Box Keyword

Used With Routines: PLOT

Corresponding System Variable: !PDT.Box

Places a box around the labels in a Date/Time axis. If you set the keyword to a value of 1, boxes are drawn around all the labels of the Date/Time axis. The default is for no boxes to be drawn.

C_Annotation Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Sets the label that will be drawn on each contour.

Usually, contours are labeled with their value. This parameter, a vector of strings, allows any text to be specified. The first label is used for the first contour drawn, and so forth. If the Levels keyword is specified, the elements of C Annotation correspond directly to the levels specified, otherwise, they correspond to the default levels chosen by the CONTOUR procedure. If there are more contour levels than elements in C Annotation, the remaining levels are labeled with their values.

Example

To produce a contour plot with three levels labeled "low", "medium", and "high":

```
CONTOUR, Z, Levels = [0.0, 0.5, 1.0], $
  C Annotation = ["low", "medium", "high"]
```

Use of this keyword implies use of the Follow keyword.

C_Charsize Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Sets the size of the characters used to annotate contour labels.

Normally, contour labels are drawn at three-fourths the size used for the axis labels (specified by the *Charsize* keyword or !P.Charsize system variable). This keyword allows the contour label size to be specified independently. Use of this keyword implies use of the *Follow* keyword.

C_Colors Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

A vector of color indices used to set the color index used to draw each contour.

This parameter is a vector, converted to integer type if necessary. If there are more contour levels than elements in *C_Colors*, the elements of the color vector are cyclically repeated.

Example

If C_Colors contains three elements, and there are seven contour levels to be drawn, the colors c_0 , c_1 , c_2 , c_0 , c_1 , c_2 , c_0 will be used for the seven levels. To call CONTOUR and set the colors to [100, 150, 200]:

```
CONTOUR, Z, C Colors = [100, 150, 200]
```

Channel Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE SURF, SURFACE, XYOUTS

Corresponding System Variable: None.

Specifies the destination channel index or mask for the operation. This parameter is used only with devices that have multiple display channels. The default is zero.

Charsize Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: !P.Charsize

Sets the overall character size for the annotation. A Charsize of 1.0 is normal. The size of the annotation on the axes may be set, relative to Charsize, with XCharsize, YCharsize, and ZCharsize. The main title is written with a character size of 1.25 times this parameter.

Charthick Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE, XYOUTS

Corresponding System Variable: !P.Charthick

Sets the thickness of characters drawn with the software fonts. Normal thickness is 1.0, double thickness is 2.0, etc. If this keyword is omitted, the value of the system variable !P.Charthick is used.

C Labels Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Specifies which contour levels should be labeled. By default, every other contour level is labeled.

C_Labels allows you to override this default and explicitly specify the levels to label. This parameter is a vector, converted to integer type if necessary. If the *Levels* keyword is specified, the elements of *C_Labels* correspond directly to the levels specified, otherwise, they correspond to the default levels chosen by the CONTOUR procedure. Setting an element of the vector to zero causes that contour label to not be labeled. A nonzero value forces labeling.

Example

To produce a contour plot with four levels where all but the third level is labeled:

Use of this keyword implies use of the Follow keyword.

C_Linestyle Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Specifies the linestyle used to draw each contour.

As with *C_Colors*, *C_Linestyle* is a vector of linestyle indices. If there are more contour levels than linestyles, the linestyles are cyclically repeated. The following table lists the available linestyles and their keyword indices:

Index	Linestyle	
0	Solid	
1	Dotted	
2	Dashed	
3	Dash Dot	
4	Dash Dot Dot	
5	Long Dashes	

Note //

The current contouring algorithm draws all the contours in each cell, rather than following contours. Hence, some of the more complicated linestyles will not be suitable for some applications.

Example

To produce a contour plot, with the contour levels directly specified in a vector V, with all negative contours drawn with dotted lines, and with positive levels in solid lines:

CONTOUR, Z, Levels = V, C_Linestyle = V LT 0.0

Clip Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: !P.Clip

Sets the coordinates of a rectangle used to clip the graphics output.

The rectangle is specified as a vector of the form $[X_0, Y_0, X_1, Y_1]$, giving coordinates of the lower-left and upper-right corners, respectively. Coordinates are specified in data coordinate units unless an overriding coordinate keyword is present, such as Normal or Device.



Even if no clipping rectangle is specified, PV-WAVE will clip to the plotting window unless the *Noclip* keyword is specified.

Example

To draw a vector using normalized coordinates with its contents clipped within a rectangle covering the display's upper-left quadrant:

PLOTS, X, Y, Clip = [0.0, 0.5, 0.5, 1.0], /Normal

Color Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE_SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: !P.Color

Sets the color index of text, lines, solid polygon fill, data, axes, and annotation. If this keyword is omitted, !P.Color specifies the color index.

Compress Keyword

Used With Routines: PLOT OPLOT

Corresponding System Variable: !PDT.Compress

Compresses out weekends and holidays from a Date/Time axis. Before you can use this keyword, you must define holidays or weekends with the procedures CREATE_HOLIDAYS and CREATE_WEEKENDS.

C_Thick Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Specifies the line thickness of lines used to draw each contour level. As with *C_Colors*, *C_Thick* is a vector of line thickness values, although the values are floating-point. If there are more contours than thickness elements, elements are repeated. If omitted, the overall line thickness specified by the *Thick* keyword parameter or !P.Thick is used for all contours.

Data Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE_SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: None.

A keyword flag, which if present, indicates that the coordinates are specified in data coordinates (the default). When used with AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, and SURFACE, this keyword also specifies that the Position and Clip coordinates are in data units.

Device Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: None.

Express coordinates in device coordinates. When used with AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, and SURFACE, this keyword also specifies that the Position and Clip coordinates are in device units.

Example

The following code displays an image contained in the variable A and then draws a contour plot of pixels (100:499, 100:399) registered over the pixels:

```
TV, A
   Display the image.
CONTOUR, A(100:499, 100:399), Position = $
    [100,100, 499,399], /Device, /Noerase, $
   XStyle = 1, YStyle = 1
        Draw the contour plot, specify the coordinates of the plot,
        in device coordinates, do not erase, set the X and Y axis
```

Note that in the above example, the keyword specification /Device is equivalent to Device = 1.

DT Range Keyword

Used With Routines: PLOT

styles to EXACT.

Corresponding System Variable: !PDT.DT_Range

Sets an exact range of values in a Date/Time axis. You must specify the desired start and end values from a Date/Time Julian value. The range may be adjusted slightly by the PLOT procedure, depending on the data. To obtain an exact range set the XStyle plotting keyword to two. For more information, see XStyle.

Fill_Pattern Keyword

Used With Routines: PLOTS, POLYFILL

Corresponding System Variable: None.

The hardware-dependent fill pattern index for the POLYFILL and PLOTS procedures. If omitted or set to 0, a solid fill results.

Follow Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

If present and nonzero, forces the CONTOUR procedure to use the line-following method instead of the cell-drawing method.

CONTOUR can draw contours using one of two different methods:

- The cell-drawing method, used by default, examines each array cell and draws all contours emanating from that cell before proceeding to the next cell. This method is efficient in terms of computer resources but does not allow contour labeling.
- The line-following method searches for each contour line and then follows the line until it reaches a boundary or closes. This method gives better looking results with dashed linestyles, and allows contour labeling, but requires more computer time. It is used if any of the following keywords is specified: C_Annotation, C_Charsize, C_Labels, Follow, or Path_Filename.

Although these two methods both draw correct contour maps, differences in their algorithms can cause small differences in the resulting plot.

Font Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: !P.Font

An integer that specifies the graphics text font index.

Font index -1 selects the software fonts, which are drawn using vectors. Font number 0 selects the hardware font of the output device. See Chapter 9, Software Fonts, in the PV-WAVE User's Guide for a complete description of the software fonts. See Appendix A, Output Devices and Window Systems, in the PV-WAVE User's Guide for more information on the hardware fonts available with each supported output device.

When using three-dimensional transformations, always use the software fonts, as the hardware fonts are not properly projected from three to two dimensions.

Gridstyle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !P.Gridstyle

Lets you change the linestyle of tick marks. The default is a solid line. Other linestyle choices and their index values are listed in the following table:

Index	Linestyle
0	Solid
1	Dotted
2	Dashed
3	Dash Dot
4	Dash Dot Dot
5	Long Dashes

One possible use for this keyword is to create an evenly spaced grid consisting of dotted or dashed lines across your plot region. To do this, first set the *Ticklen* keyword to 0.5. This ensures that the dotted or dashed tick style will appear correctly on your plot. Then set the *Gridstyle* keyword to the style you want to use. For example:

PLOT, mydata, Ticklen = 0.5, Gridstyle = 1 produces a plot with a dotted grid across the entire plot region.

See also Example 2: Specifying Tick Lengths on page 80 of the PV-WAVE User's Guide.

Horizontal Keyword

Used With Routines: SURFACE

Corresponding System Variable: None.

If set, causes SURFACE to only draw lines across the plot perpendicular to the line of sight. The default is for SURFACE to draw both across the plot and from front to back.

Levels Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Specifies a vector containing the contour levels (maximum of 150) drawn by the CONTOUR procedure.

A contour is drawn for each level specified in *Levels*. If omitted, the data range is divided into approximately six equally-spaced levels.

Example

To draw a contour plot with levels at 1, 100, 1000, and 10000:

```
CONTOUR, Z, Levels = [1, 100, 1000, 10000]
```

To draw a contour plot with levels at 50, 60, ..., 90, 100:

CONTOUR, Z, Levels = FINDGEN(6) * 10 + 50

Line Fill Keyword

Used With Routines: PLOTS, POLYFILL

Corresponding System Variable: None.

Indicates that polygons are to be filled with parallel lines, rather than using solid or patterned filling methods.

When using the line-drawing method of filling, the thickness, linestyle, orientation, and spacing of the lines may be specified with keywords.

Linestyle Keyword

Used With Routines: OPLOT, PLOT, PLOTS, POLYFILL, **SURFACE**

Corresponding System Variable: !P.Linestyle

Specifies the linestyle used to draw the lines or connect data points. The linestyle index is an integer, as shown in the following table:

Index	Linestyle	
0	Solid	
1	Dotted	
2	Dashed	
3	Dash Dot	
4	Dash Dot Dot	
5	Long Dashes	

Lower_Only Keyword

Used With Routines: SURFACE

Corresponding System Variable: None.

Indicates that only the lower surface of the object is to be drawn.

Max_Levels Keyword

Used With Routines: PLOT

Corresponding System Variable: !PDT.Max Levels

Sets the maximum number of levels on a Date/Time axis. For example, assume that the Date/Time data contains years, months, days, hours, minutes, and seconds. If this keyword is set to three, then the Date/Time axis will show three levels: seconds, minutes, and hours.

Max_Value Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Data points with values equal to or above this value are ignored when contouring. Cells containing one or more corners with values above

Max Value will have no contours drawn through them.

Month_Abbr Keyword

Used With Routines: PLOT

Corresponding System Variable: !PDT.Month Abbr

Abbreviates month or quarter names to either three characters or one character, depending on the space available on the Date/Time axis. If a one-character abbreviation does not fit, no label is used. If the complete label can fit, it is not abbreviated, even if the keyword is specified. Month names are specified in the system variable !Month_Names. Quarter names are specified in the !Quarter_Names system variable.

NLevels Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

The number of equally-spaced contour levels that are produced by CONTOUR. The maximum is 150. The default is six.

If the Levels parameter, which explicitly specifies the value of the contour levels, is present this keyword has no effect. If neither parameter is present approximately six levels are drawn.

If the minimum and maximum Z values are Z_{min} and Z_{max} , then the value of the ith level is:

$$Z_{min} + (i + 1)(Z_{max} - Z_{min})/(NLevels + 1)$$

where i ranges from 0 to NLevels - 1.

Noclip Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE SURF, SHADE SURF IRR, SURFACE, XYOUTS

Corresponding System Variable: !P.Noclip

Suppresses clipping of the plot, lines, and vector-drawn text. Use this keyword if you want to write text outside the plot window. By default the plot is clipped within the plotting window.

Nodata Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF_IRR, SURFACE

Corresponding System Variable: None.

If this keyword is set, only the axes, titles, and annotation are drawn. No data points are plotted.

Example

To draw an empty set of axes between some given values:

PLOT, [XMIN, XMAX], [YMIN, YMAX], /Nodata

Noerase Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: None.

Specifies that the screen or page is not to be erased. By default the screen is erased, or a new page is begun, before a plot is produced.

Normal Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE_SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: None.

Indicates that the coordinates are in the normalized coordinate system and range from 0.0 to 1.0.

When used with AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, and SURFACE, indicates that the *Clip* and/or *Position* coordinates are in the normalized coordinate system and range from 0.0 to 1.0.

Nsum Keyword

Used With Routines: OPLOT, PLOT

Corresponding System Variable: None.

Indicates the number of data points to average when plotting.

If Nsum is larger than 1, every group of Nsum points is averaged to produce one plotted point. If there are m data points, then m / Nsum points are displayed. On logarithmic axes a geometric average is performed.

It is convenient to use *Nsum* when there is an extremely large number of data points to plot because it plots fewer points, the graph is less cluttered, and it is quicker.

Orientation Keyword

Used With Routines: PLOTS, POLYFILL, XYOUTS

Corresponding System Variable: None.

Specifies the angle in degrees, counterclockwise from horizontal, of the text baseline and the lines used to fill polygons.

When used with the POLYFILL procedure, this keyword forces the Linestyle type of fill, rather than solid or patterned fill.

Overplot Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Indicates the CONTOUR procedure is to overplot.

No axes are drawn and the previously established scaling remains in effect. You must explicitly specify the values of the contour levels with the Levels keyword when using this option.

Path Filename Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Specifies the name of a file to contain the contour positions.

If Path Filename is present, CONTOUR does not draw the contours, but rather, opens the specified file and writes the positions, in normalized coordinates, into it. The file consists of a series of logical records containing binary data. Each record is preceded with a header structure defining the contour as follows:

```
{CONTOUR HEADER, TYPE : OB, HIGH : OB, $
  LEVEL: 0, NUM: OL, VALUE: 0.0}
```

The fields are:

TYPE - A byte which is zero if the contour is open, and one if it is closed.

- HIGH A byte which is 1 if the contour is closed and above its surroundings, and is 0 if the contour is below. This field is meaningless if the contour is not closed.
- LEVEL A short integer with value greater than or equal to zero. (It is an index into the *Levels* array).
- NUM The longword number of data points in the contour.
- VALUE The contour value. This a single-precision floating-point value.

Following the header in each record are NUM pairs of single-precision floating (X,Y) values, expressed in normalized coordinates.

The POLYCONTOUR procedure can be used along with this file to fill the closed contours with specified colors. A typical application is to use CONTOUR with the *Path_Filename* keyword to get the path information, use POLYCONTOUR to fill the closed contours, and then use CONTOUR with the *Noerase* keyword to overplot the contours.

Use of this keyword implies use of the Follow keyword.

Pattern Keyword

Used With Routines: PLOTS, POLYFILL

Corresponding System Variable: None.

A rectangular array of pixels giving the fill pattern.

If this keyword parameter is omitted, POLYFILL fills the area with a solid color. The pattern array may be of any size; if it is smaller than the filled area the pattern array is cyclically repeated.

Example

To fill the current plot window with a grid of dots:

```
Pattern = BYTARR(10, 10)
Define pattern array as 10-by-10.
```

Pattern(5,5) = 255Set center pixel to bright.

```
POLYFILL, !X.Window([0, 1, 1, 0]), $
   !Y.Window([0, 0, 1, 1]), /Normal, $
  Pattern = Pattern
```

Fill the rectangle defined by the four corners of the window with the pattern.

Polar Keyword

Used With Routines: OPLOT, PLOT

Corresponding System Variable: None.

Polar plots are produced when this keyword is present and nonzero.

The X and Y vector parameters, both of which must be present, are first converted from polar to cartesian coordinates. The first parameter is the radius, and the second is θ , expressed in radians.

To make a polar plot:

PLOT, /Polar, R, THETA

Position Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF_IRR, SURFACE

Corresponding System Variable: !P.Position

Allows direct specification of the plot window.

Position is a four-element vector giving, in order, the coordinates $[(X_0, Y_0), (X_1, Y_1)]$ of the lower-left and upper-right corners of the data window. Coordinates are expressed in normalized units ranging from 0.0 to 1.0, unless the keyword /Device is present, in which case they are in actual device units.

When setting the position of the window, be sure to allow space for the annotation, which resides outside the window. PV-WAVE outputs the message

%, Warning: Plot truncated.

if the plot region is larger than the screen or page size. The plot region is the rectangle enclosing the plot window and the annotation.

When plotting in three dimensions, the *Position* keyword is a sixelement vector with the first four elements describing, as above, the XY position, and with the last two elements giving the minimum and maximum Z coordinates. The Z specification is always in normalized coordinate units.

When making more than one plot per page it is more convenient to set !P.Multi than to manipulate the position of the plot directly with the *Position* keyword.

Example

The following statement produces a contour plot with data plotted in only the upper-left quarter of the screen:

```
CONTOUR, Z, Position = [0.0, 0.5, 0.5, 1.0]
```

Because no space on the left or top edges was allowed for the axes or their annotation, the warning message described above results.

Psym Keyword

Used With Routines: OPLOT, PLOT, PLOTS, POLYFILL

Corresponding System Variable: !P.Psym

Specifies the symbol used to mark each data point.

Normally, *Psym* is 0, data points are connected by lines, and no symbols are drawn to mark the points. Specify this keyword to mark data points with symbols. (Alternatively, , you can set !P.Psym to the symbol index as given in Table 4-4 on page 548.) The keyword *Symsize* is used to set the size of the symbols.

Negative values of Psym cause the symbol designated by |Psym| to be plotted at each point with solid lines connecting the symbols. For example, a Psym value of -5 plots triangles at each data point and connects the points with lines.

Example

The following PV-WAVE code plots an array using points, and then overplots the smoothed array, connecting the points with lines:

```
PLOT, A, Psym = 3
   Plot using points.
OPLOT, SMOOTH(A, 7)
   Overplot smoothed data.
```

Save Keyword

Used With Routines: AXIS, SHADE_SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: None.

Saves the 3D to 2D transformation matrix established by SURFACE and SHADE SURF, and specified by the Ax and Az keywords, in the system variable field !P.T.

Use this keyword when combining the output of SURFACE and SHADE SURF with the output of other routines in the same plot.

When used with AXIS, the Save keyword parameter saves the scaling parameters established by the call back in the appropriate axis system variable, !X, !Y, or !Z. This causes subsequent overplots to be scaled to the new axis.

Example

To display a two-dimensional array using SURFACE, and to then superimpose contours over the surface: (This example assumes that !P.T3d is zero, its default value.)

```
SURFACE, Z, /Save
   Make a surface plot and save the transformation.
```

```
CONTOUR, Z, /Noerase, /T3d
   Make contours, don't erase, use the 3D to 2D transform placed
   in !P.T by SURFACE.
```

To display a surface and to then display a flat contour plot, registered above the surface:

SURFACE, Z, /Save

Make the surface, save transform.

CONTOUR, Z, /Noerase, /T3d, ZValue = 1.0 Now display a flat contour plot, at the maximum Z value (normalized coordinates). You can display the contour plot below the surface with a ZValue of 0.0.

Size Keyword

Used With Routines: XYOUTS

Corresponding System Variable: None.

Specifies the character size as a factor of the normal character size. Normal size is 1.0.

Skirt Keyword

Used With Routines: SURFACE

Corresponding System Variable: None.

A skirt around the array at a given Z value is drawn if this keyword parameter is present. The Z value is expressed in data units.

For example:

SURFACE, A, Skirt = 100

draws the surface of A with a skirt at the Z value 100.

If the skirt is drawn, each point on the four edges of the surface is connected to a point on the skirt which has the given Z value, and the same X and Y values as the edge point. In addition, each point on the skirt is connected to its neighbor.

Spacing Keyword

Used With Routines: PLOTS, POLYFILL

Corresponding System Variable: None.

Specifies the spacing, in centimeters, between the parallel lines used to fill polygons.

Spline Keyword

Used With Routines: CONTOUR

Corresponding System Variable: None.

Specifies that contour paths are to be interpolated using cubic splines.

Use of this keyword implies the use of the Follow keyword. The appearance of contour plots of arrays with low resolution may be improved by using spline interpolation. In rare cases, contour lines that are close together may cross because of interpolation.

Splines are especially useful with small data sets (less than 15 array dimensions). With larger data sets the smoothing is not as noticeable and the expense of splines increases rapidly with the number of data points.

You may specify the length of each interpolated line segment in normalized coordinates by including a value with this keyword. The default value is 0.005 which is obtained when the parameter /Spline is present. Smaller values for this parameter yield smoother lines, up to the resolution of the output device, at the expense of more computations.

Start Level Keyword

Used With Routines: PLOT

Corresponding System Variable: !PDT.Start Level

Specifies the initial level of tick labels to be displayed on a Date/ Time axis. The subsequent levels depend on the data range and the first level selected.

Index values for the Start Level keyword are listed below:

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

Subtitle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !P.Subtitle

Produces a subtitle underneath the X axis containing the text in this string parameter.

Symsize Keyword

Used With Routines: OPLOT, PLOT, PLOTS, POLYFILL

Corresponding System Variable: None.

Specifies the size of the symbols drawn when *Psym* is set. The default size of 1.0 produces symbols approximately the same size as a character.

T3d Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, PLOTS, POLYFILL, SHADE_SURF, SHADE_SURF_IRR, SURFACE, XYOUTS

Corresponding System Variable: !P.T3d

A keyword flag which, if present, indicates that the generalized transformation matrix in !P.T is to be used.

!P.T must contain a valid transformation matrix before the T3d keyword can be used. The matrix can be set by using the Save plotting keyword with the appropriate plotting routine.

If T3d is not present the user-supplied coordinates are simply scaled to screen coordinates. See the examples in the description of the Save plotting keyword.

A valid transformation matrix can be placed in !P.T in several ways:

Use the Save keyword to save the transformation matrix from
an earlier graphics operation.

- ☐ Establish a transformation matrix using the T3D user library procedure.
- ☐ Set the value of !P.T directly.

Text Axes Keyword

Used With Routines: XYOUTS

Corresponding System Variable: None.

Specifies the plane of vector-drawn text when three-dimensional plotting is enabled. By default, text is drawn in the plane of the XY axes. The horizontal text direction is in the X plane, and the vertical text direction is in the Y plane.

Values of this keyword may range from 0 to 5, with the following effect: 0 for XY, 1 for XZ, 2 for YZ, 3 for YX, 4 for ZX, and 5 for ZY. The notation ZY means that the horizontal direction of the text lies in the Z plane, and the vertical direction of the text is drawn in the Y plane.

Thick Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !P.Thick

Controls the thickness of the lines connecting points. A thickness of 1.0 is normal, 2.0 is double-wide, etc.

Tickformat Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !P.Tickformat

Lets you use FORTRAN-style format specifiers to change the format of tick labels on the X, Y, and Z axes. For example:

```
PLOT, mydata, Tickformat = '(F5.2)'
```

The resulting plot's tick labels are formatted with a total width of five characters carried to two decimal places. As expected, the width field expands automatically to accommodate larger values. For more information on format specifiers, see Explicitly Formatted Input and Output on page 165 of the PV-WAVE Programmer's Guide. See also Example 3: Specifying Tick Label Formats on page 82 of the PV-WAVE User's Guide.

Note that only the I (integer), F (floating-point), and E (scientific notation) format specifiers can be used with Tickformat. Also, you cannot place a quoted string inside a tick format. For example, ("<", F5.2, ">") is an invalid *Tickformat* specification.

See also XTickformat, YTickformat, and ZTickformat.

Ticklen Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !P.Ticklen

Controls the length of the axis tick marks, expressed as a fraction of the window size. The default value is 0.02. Ticklen of 0.5 produces a grid, while a negative Ticklen makes tick marks that extend outside the plot region, rather than inwards.

Example

To produce outward-going tick marks of the normal length:

```
PLOT, X, Y, Ticklen = -0.02
```

To provide a new default tick length, set the system variable !P.Ticklen.

Title Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SURFACE

Corresponding System Variable: !P.Title

Produces a main title centered above the plot window.

The text size of this main title is larger than the other text by a factor of 1.25.

For example:

PLOT, X, Y, Title = 'Final Results'

Upper Only Keyword

Used With Routines: SURFACE

Corresponding System Variable: None.

Indicates that only the upper surface of the object is to be drawn. By default, both surfaces are drawn.

Week Boundary Keyword

Used With Routines: PLOT

Corresponding System Variable: !PDT.Week_Boundary

Sets the day of the week on which week tick marks are drawn for the week level on a Date/Time axis.

For example, if you set the week boundary to 'Sunday', weekly tick marks are drawn for each Sunday and each one is labeled with the date.

The week boundary index can be one of the following integers:

Index	Day of Week	
0	Sunday	
1 (the default)	Monday	
2	Tuesday	
3	Wednesday	
4	Thursday	
5	Friday	
6	Saturday	

Width Keyword

Used With Routines: XYOUTS

Corresponding System Variable: None.

Returns the width of the text string, in normalized coordinate units, to the designated variable.

For example, to put the width of a text string in the variable w_title, use:

XYOUTS, x, y, 'Title of Graph', Width = w title

XAxis Keyword

Used With Routines: AXIS

Corresponding System Variable: None.

The XAxis and YAxis keywords indicate which type of axis is to be drawn by the AXIS procedure and its placement.

See also YAxis and ZAxis.

XCharsize Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !X.Charsize

The size of the characters used to annotate the axis and its title.

This field is a scale factor applied to the global scale factor set by !P.Charsize or the keyword *Charsize*.

See also Charsize, YCharsize, and ZCharsize.

XGridstyle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !X.Gridstyle

Lets you change the linestyle of tick marks on the X axis. The default is a solid line.

See also Gridstyle.

XMargin Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !X.Margin

A two-element array specifying the margin on the left (bottom) and right (top) sides of the plot window, in units of character size. Default margins are 10 and 3 for the X axis, and 4 and 2 for the Y axis.

See also YMargin and ZMargin.

XMinor Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !X.Minor

The number of minor tick intervals on the particular axis. If set to 0, the default, PV-WAVE automatically determines the number of minor ticks in each major tick mark interval. Setting this parameter to -1 suppresses the minor ticks, and setting it to a positive, nonzero number n produces n minor tick intervals, and n-1 minor tick marks.

See also YMinor and ZMinor.

XRange Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !X.Range

The desired data range of the axis, a two-element vector. The first element is the axis minimum, and the second is the maximum. PV-WAVE will frequently round this range.

See also YRange and ZRange.

XStyle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !X.Style

Allows specification of axis options such as rounding of tick values and selection of a box axis. Each option is encoded in a bit. See the following table for details:

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 this bit is set, the axes are extended by 5% in each direction, leaving a border around the data.
2	4	None. If this bit is set, the axis and its text is 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 or right axis.
4	16	Inhibits setting the Y axis minimum value to zero, when the data are all positive and nonzero. The keyword <i>YNozero</i> sets this bit temporarily.

See also YStyle and ZStyle.

XTickformat Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !X.Tickformat

Lets you use FORTRAN-style format specifiers to change the format of tick labels for the X axis. For example:

```
PLOT, mydata, XTickformat = '(F5.2)'
```

This keyword works basically the same way as the Tickformat keyword. See Tickformat Keyword on page 522 for more information.

See also Tickformat, YTickformat, and ZTickformat.

XTicklen Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !X.Ticklen

Functions the same as the keyword Ticklen. XTicklen, however, can be applied to the X axis. XTicklen supersedes the value of the Ticklen setting.

See also Ticklen, YTicklen, and ZTicklen.

XTickname Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !X.Tickname

A string array, of up to 30 elements, containing the annotation of each major tick mark.

If omitted, or if a given string element that contains the null string, PV-WAVE labels the tick mark with its value. To suppress the tick label, supply a string array of one-character-long blank strings. You can do this with the command:

(Null strings cause PV-WAVE to number the tick mark with its value.) Note that if there are n tick mark intervals, there are n + 1 tick marks and labels.

See also YTickname and ZTickname.

XTicks Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !X.Ticks

The number of major tick *intervals* to draw for the axis. If omitted PV-WAVE will select from three to six tick intervals. Setting this field to n, where n > 0, produces exactly n tick intervals, and n + 1 tick marks.

See also YTicks and ZTicks.

XTickv Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !X.Tickv

The data values for each tick mark, an array of up to 30 elements.

This keyword allows you to directly specify tick data values, producing graphs with non-linear tick marks. PV-WAVE scales the axis from the first tick value to the last, unless you directly specify a range. If you specify n tick intervals, you must specify n + 1 tick values.

See also YTicky and ZTicky.

XTitle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !X.Title

Places a title below the X axis.

See also Title, YTitle, and ZTitle.

XType Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT

Corresponding System Variable: None.

Specifies a linear axis if zero; specifies a logarithmic axis if one; and if set to 2, enables compressed Julian numbers to be used directly with the PLOT or OPLOT procedures.

YAxis Keyword

Used With Routines: AXIS

Corresponding System Variable: None.

This keyword works like XAxis.

YCharsize Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Y.Charsize

This keyword works like XCharsize.

YGridstyle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !Y.Gridstyle

Lets you change the linestyle of tick marks on the Y axis. The default is a solid line.

For more information, see Gridstyle.

YMargin Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Y.Margin.

This keyword works like XMargin.

YMinor Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Y.Minor.

This keyword works like XMinor.

YNozero Keyword

Used With Routines: AXIS, OPLOT, PLOT

Corresponding System Variable: None.

Inhibits setting the minimum Y axis value to zero when the Y data are all positive and nonzero, and no explicit minimum Y value is specified (using *Yrange*, or !Y.Range).

By default, the Y axis spans the range of 0 to the maximum value of Y, in the case of positive Y data. Set bit 4 in !Y.Style to make this option the default.

YRange Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Y.Range

This keyword works like XRange.

YStyle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Y.Style

This keyword works like XStyle.

YTickformat Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !Y.Tickformat

Lets you use FORTRAN-style format specifiers to change the format of tick labels for the Y axis. For example:

PLOT, mydata, YTickformat = '(F5.2)'

This keyword works like XTickformat.

YTicklen Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !Y.Ticklen

This keyword works like XTicklen.

YTickname Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Y.Tickname

This keyword works like XTickname.

YTicks Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !Y.Ticks

This keyword works like XTicks.

YTickv Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Y.Tickv

This keyword works like XTickv.

YTitle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !Y.Title

This keyword works like XTitle.

YType Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT Corresponding System Variable: None.

This keyword works like XType, except this keyword can have no effect on Date/Time plots.

Z Keyword

Used With Routines: PLOTS, POLYFILL, XYOUTS

Corresponding System Variable: None.

Provides the Z coordinate if a Z parameter is not present in the call. This is of use only if the three-dimensional transformation is in effect.

ZAxis Keyword

Used With Routines: AXIS, CONTOUR, SURFACE

Corresponding System Variable: None.

Specifies the existence of a Z axis for CONTOUR, and the placement of the Z axis for SURFACE. For AXIS, the ZAxis keyword indicates that a Z axis is to be drawn and where it should be placed.

CONTOUR draws no Z axis by default. Include the ZAxis keyword in the call to CONTOUR to draw a Z axis. This is of use only if a three-dimensional transformation is established.

By default, SURFACE draws the Z axis at the upper-left corner of the axis box. To suppress the Z axis, use ZAxis = -1 in the call. The position of the Z axis is determined from ZAxis as follows: 1 = lower-right, 2 = lower-left, 3 = upper-left, and 4 = upper-right.

ZCharsize Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !Z.Charsize

This keyword works like XCharsize.

ZGridstyle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !Z.Gridstyle

Lets you change the linestyle of tick marks on the Z axis. The default is a solid line.

For more information, see Gridstyle.

ZMargin Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Z.Margin

This keyword works like XMargin.

ZMinor Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Z.Minor

This keyword works like XMinor.

ZRange Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Z.Range

This keyword works like XRange.

ZStyle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Z.Style

This keyword works like XStyle, except this keyword has no effect on Date/Time plots.

ZTickformat Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !Z.Tickformat

This keyword works like XTickformat.

ZTicklen Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SURFACE

Corresponding System Variable: !Z.Ticklen

This keyword works like XTicklen.

ZTickname Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: !Z.Tickname

This keyword works like XTickname.

ZTicks Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Z.Ticks

This keyword works like XTicks.

ZTickv Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Z.Tickv

This keyword works like XTickv.

ZTitle Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE SURF, SHADE_SURF_IRR, SURFACE

Corresponding System Variable: !Z.Title

This keyword works like XTitle.

ZType Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT

Corresponding System Variable: None.

This keyword works like XType.

ZValue Keyword

Used With Routines: AXIS, CONTOUR, OPLOT, PLOT, SHADE_SURF, SHADE SURF IRR, SURFACE

Corresponding System Variable: None.

Sets the Z coordinate, in normalized coordinates in the range of 0 to 1, of the axis and data output from PLOT, OPLOT, and CONTOUR.

This has an effect only if !P.T3d is set and the three-dimensional to two-dimensional transformation is stored in !P.T. If *ZValue* is not specified, CONTOUR will output each contour at its Z coordinate, and the axes and title at a Z coordinate of 0.0.

System Variables

This chapter discusses the PV-WAVE system variables. For more information on system variables, see Chapter 2, Constants and Variables, in the PV-WAVE Programmer's Guide.

!C

The cursor system variable. Currently, its only function is to contain the subscript of the largest or smallest element found by the MAX and MIN functions.

!D

A structure containing information about the current graphics output device. Fields are described in the following sections.

The fields of the !D structure are read-only.

!D.Fill_Dist

The line interval, in device coordinates, required to obtain a solid fill.

!D.Flags

A longword of flags. Each bit is a flag encoded as shown in the following table:

Table 4-1: !D.Flags Bit Definitions

Bit	Value	Function
0	1	Device has scalable pixel size (e.g., Post-Script).
1	2	Device can output text at an arbitrary angle using hardware.
2	4	Device can control line thickness with 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 (TVRD).
8	256	Device supports windows.

!D.N_Colors

The number of simultaneously available colors. In the case of devices with windows, this field is set after the window is initialized. For monochrome systems, !D.N_Colors is 2, and for color systems it is normally 256.

!D.Name

A string containing the name of the device.

!D.Table Size

This field contains the number of color table indices available on the device. Devices without color tables have this field set to 0.

!D.Unit

The logical number of the file open for output by the current graphics device. This field only has meaning if the file is accessible to the user from PV-WAVE, and is 0 if no file is open. For example, the PostScript driver maintains this field with the unit number of the file open for PostScript output. In the case of Tektronix output to a file, !D.Unit may be set to either + or – the logical unit number.

!D.Window

The index of the currently open window. Set to -1 if no window is open. Used only with devices that support windows.

!D.X_Ch_Size / !D.Y_Ch_Size

The normal width and height of a character in device units. These fields are set after the window is initialized.

$!D.X_Px_Cm / !D.Y_Px_Cm$

The number of pixels per centimeter in the X and Y directions.

!D.X_Size / !D.Y_Size

The total size of the display in the X and Y directions. These values determine the maximum possible size of any PV-WAVE

window, unless !D.X_Vsize and !D.Y_Vsize are smaller, in which case, !D.X_Vsize and !D.Y_Vsize determine the maximum size.

!D.X Vsize / !D.Y Vsize

Reports the size of the current window, and is updated when the window is resized.

!Date_Separator

A string containing, by default, a slash (/) character. This character is used to separate the parts of a date on output (for example, 5/15/1992). To use a different character, change the value of this variable. This variable is used by the DT_PRINT, DT_TO_STR, and DC_WRITE * routines.

!Day_Names

An array of strings containing the names of the days of the week. This system variable is used by the DAYNAME function to return the names of the days for a specified PV-WAVE date and time variable.

!Dir

Contains the name of the main PV-WAVE directory (that is, the directory containing the files PV-WAVE needs to run).

!Display_Size

Contains the pixel dimensions of the display screen for devices running the X Window System.

!Dpi

Contains the double-precision value of pi. This is a read-only system variable.

!DT Base

Contains the value of Julian Day 1 (September 14, 1752) as a !DT structure. Used in various Date/Time calculations. This value can be overridden using the *Base* keyword as a parameter to the SEC TO DT and DT TO SEC routines.

This variable can also be modified directly; however, if you do this, you must set the last field (the recalc flag) of the !DT structure to 1. For more information, see the section Recalc Flag in Chapter 7, Working with Date/Time Data, in the PV-WAVE User's Guide.

!Dtor

Contains the conversion factor to convert degrees to radians. The value is π / 180, which is approximately 0.01745. This is a read-only system variable.

!Edit_Input

Enables or disables the keyboard line-editing feature.

!Err

Contains the code of the last I/O error message. The CURSOR function also uses !Err to store the return value of the function. This enables the user to determine which mouse button was pushed.

!Err_String

Contains the text of the last I/O error message. This is a read-only system variable.

!Holiday_List

This system variable is created by the CREATE_HOLIDAYS procedure. It is a Date/Time variable containing holidays as defined by CREATE_HOLIDAYS. This system variable does not have a default value. This variable can hold up to 50 holiday definitions.

!Journal

Contains the logical unit number of journal output. If there is no journal output, !Journal = 0. This is a read-only system variable.

!Lang

Identifies the language currently being used. The default is "american".

!Month Names

An array of strings containing the names of the months. This system variable is used by the MONTH_NAME function to return the names of the months for a specified PV-WAVE date and time variable.

!Mouse

Used by the CURSOR function to store the X and Y position of the mouse, the mouse button status, and a date/time stamp. The fields for this variable are:

!Mouse.X

!Mouse.Y

!Mouse.Button

!Mouse.Time

For additional information on this variable and its fields, see the CURSOR procedure..

!Msg_Prefix

Contains a prefix string for error messages issued by PV-WAVE. The default is a percent sign. This system variable provides a way to distinguish error messages from normal output.

!Order

Controls the direction of image transfers. If !Order is 0, images are transferred from bottom to top (i.e., the row with a 0 subscript is

written on the bottom). Setting !Order to 1 transfers images from top to bottom.

!P

The main system variable structure for plotting.

All fields, except !P.Multi, have a directly corresponding keyword parameter in the main plot procedures: AXIS, PLOT, OPLOT, CONTOUR, and SURFACE.

The fields of the !P structure are explained in the following sections.

!P.Background

Corresponding Plot Keyword: Background

The background color index. When erasing a window, all pixels are set to this color. The default value is 0.

!P.Charsize

Corresponding Plot Keyword: Charsize

The overall character size of all annotation. The normal size is 1. The main plot title size is 1.25 times this parameter.

!P.Charthick

Corresponding Plot Keyword: Charthick

The thickness of characters drawn with the vector fonts. Normal thickness is 1.0, double thickness is 2.0, and so on.

!P.Clip

Corresponding Plot Keyword: Clip

The device coordinates of a rectangle used to clip the graphics window. The rectangle is specified as a four-element vector of the form $[(X_0, Y_0), (X_1, Y_1)]$. The coordinates specify the lower-left

and upper-right corners of the clipping rectangle, respectively. Normally the clipping rectangle is set to the plot window.

!P.Color

Corresponding Plot Keyword: Color

Color index used to draw data, axes, and annotation.

!P.Font

Corresponding Plot Keyword: Font

The index of the graphics text font. Font = -1 uses the software-drawn fonts (also called Hershey or vector-drawn fonts). Font = 0 uses the hardware-drawn fonts. See *Software Fonts* on page 291 of the *PV*=*WAVE User's Guide*, for a complete description of the vector-drawn fonts. For information on hardware-drawn fonts available for a particular output device, see Appendix A, *Output Devices and Window Systems*, in the *PV*=*WAVE User's Guide*.

When using three-dimensional transformations, always use the vector-drawn fonts, as the hardware fonts are not properly projected from three to two dimensions.

!P.Gridstyle

Corresponding Plot Keyword: Gridstyle

Lets you change the default linestyle of X, Y, and Z axis tick marks. The default is a solid line. Other linestyle choices and their index values are listed in the following table:

Table 4-2: Linestyles

Linestyle
Solid
Dotted
Dashed
Dash Dot

4	Dash Dot Dot Dot
5	Long Dashes

!P.Linestyle

Corresponding Plot Keyword: Linestyle

The style of the lines used to connect points. The linestyle index is an integer, as described in the following table:

Table 4-3: Linestyles

Index	Linestyle	
0	Solid	
1	Dotted	
2	Dashed	
3	Dash Dot	
4	Dash Dot Dot Dot	
5	Long Dashes	

!P.Multi

Allows making multiple plots on a page or a screen. It is a fiveelement integer array defined as follows:

- !P.Multi(0) contains the number of plots remaining on the page. If !P.Multi(0) is less than or equal to 0, the page is cleared, the next plot is placed the upper-left-hand corner, and !P.Multi(0) is reset to the number of plots per page.
- !P.Multi(1) is the number of plot columns per page. If ≤ 0 , one is assumed. If more than two plots are positioned in either the X or Y direction, the character size is halved.
- !P.Multi(2) is the number of rows of plots per page. If ≤ 0 , one is assumed.
- !P.Multi(3) is the number of plots stacked in the Z dimension.

• !P.Multi(4) is 0 to make plots from left to right (column major), and top to bottom, and is 1 to make plots from top to bottom, left to right (row major).

Example

To position two plots across the page:

To position two plots vertically:

To make four plots per page, two across and two up and down:

$$!P.Multi = [0, 2, 2, 0, 0]$$

and then call plot four times.

To reset !P.Multi back to the normal one plot per page:

```
!P.Multi = 0
```

For more information on !P.Multi, see *Drawing Multiple Plots on a Page* on page 83 of the *PV*-WAVE User's Guide.

!P.Noclip

Corresponding Plot Keyword: Noclip

A field, which if set, inhibits the clipping of the graphic vectors and vector-drawn text. By default, graphic vectors are clipped within the plotting window.

!P.Nsum

Corresponding Plot Keyword: Nsum

The number of adjacent points to sum to obtain a plotted point. If !P.Nsum is larger than 1, every group of !P.Nsum points is averaged to produce one plotted point. If there are m data points, then m / !P.Nsum points are displayed. On logarithmic axes a geometric average is performed.

It is convenient to use *Nsum* when there is an extremely large number of data points to plot because it plots fewer points, the graph is less cluttered, and it is quicker.

!P.Position

Corresponding Plot Keyword: Position

Specifies the position of the plot within the graphics window. It is called by specifying a four-element vector as follows:

```
!P.Position = [Xmin, Ymin, Xmax, Ymax]
```

The minimum and maximum position values are specified in normalized coordinates. For example, to position a plot in the center of the screen, type:

```
!P.Position = [0.2, 0.2, 0.8, 0.8]
```



The !P.Position variable will only position the plot itself, and the annotation on the axes may be cut off. If the plot and its associated annotation must be positioned, use the Region field of the !P system variable.

!P.Psym

Corresponding Plot Keyword: Psym

The plot symbol index. Each point drawn by PLOT and OPLOT is marked with a symbol if this field is non-zero. Set !P.Psym to the symbol index as given in Table 4-4 on page 548 to mark data points with symbols.

Negative values of !P.Psym cause the symbol designated by |!P.Psym| to be plotted at each point with solid lines connecting the symbols. For example, a !P.Psym value of -5 plots triangles at each data point and connects the points with lines.

Table 4-4: Symbols Selected by !P.Psym

!P.Psym Value	sym Value Symbol Drawn	
0	No symbol, connect points with solid lines	
1	Plus sign	
2	Asterisk	
3	Period	
4	Diamond	
5	Triangle	
6	Square	
7	X	
8	User-defined, see the USERSYM procedure	
9	Undefined	
10	Data points are plotted in the histogram mode. Horizontal and vertical lines con- nect the plotted points, as opposed to the normal method of connecting points with straight lines.	
–value	Negative values connect symbols with solid lines	

!P.Region

Specifies the positioning of the plot and its associated annotation (i.e., anything that can be specified with the PLOT command). The difference between !P.Region and !P.Position is that !P.Region provides a margin around the plot to accommodate plot annotation. It is called by specifying !P.Region's maximum and minimum values in normalized coordinates:

```
!P.Region = [Xmin, Ymin, Xmax, Ymax]
```

For example, to position a plot with its associated annotation in the center of the screen, type:

$$!P.Region = [0.2, 0.2, 0.8, 0.8]$$

!P.Subtitle

Corresponding Plot Keyword: Subtitle

The plot subtitle, placed under the X axis label.

!P.T

Contains the homogeneous 4-by-4 transformation matrix.

!P.T3D

Corresponding Plot Keyword: T3d

Enables the three-dimensional to two-dimensional transformation contained in the homogeneous 4-by-4 matrix !P.T.

!P.Thick

Corresponding Plot Keyword: Thick

The thickness of the lines connecting points. A thickness of 1.0 is normal, 2 is double-wide, and so on.

!P.Tickformat

Corresponding Plot Keyword: Tickformat

Changes the default format for X, Y, and Z axis tick labels using FORTRAN-style format specifiers.

See !X.Tickformat for more information.

!P.Ticklen

Corresponding Plot Keyword: Ticklen

The length of the tick marks, expressed as a fraction of the plot size. The default value is 0.02. A !P.Ticklen of 0.5 produces a grid, while a negative value for !P.Ticklen makes tick marks that extend outside the window, rather than inwards.

See also !X.Ticklen, !Y.Ticklen, and !Z.Ticklen.

!P.Title

Corresponding Plot Keyword: *Title*

The main plot title. The text size of this main title is larger than the other text by a factor of 1.25.

!Path

Contains the colon-separated directory path to search for procedures, functions, and arguments of executive commands.

!PDT

The main system variable structure for Date/Time plotting attributes. The fields of this structure are described below.

!PDT.Box

Corresponding Plot Keyword: Box

If zero, the background box for the Date/Time axis labels is off; if one, the background box is turned on. The default is zero — no box.

!PDT.Compress

Corresponding Plot Keyword: Compress

If zero, compression is off; if one, compression is on. The default is zero — no compression.

!PDT.Exclude Holiday

If compression is set with the Compress keyword or !PDT.Compress, and this system variable is set to one, holidays are excluded from the results of Date/Time routines such as DT ADD, DT_SUBTRACT, and DT DURATION. The default is one.

!PDT. Exclude Weekend

If compression is set with the Compress keyword or !PDT.Compress, and this system variable is set to one, weekends are excluded from the results of Date/Time routines such as DT_ADD, DT_SUBTRACT, and DT_DURATION. The default is one.

!PDT.Max Levels

Corresponding Plot Keyword: Max Levels

Sets the maximum number of levels on a Date/Time axis. For more information, see the Max Levels keyword in Chapter 3, Graphics and Plotting Keywords.

!PDT.Month Abbr

Corresponding Plot Keyword: Month_Abbr

If one, month names are abbreviated if there is not enough room to draw them on a plot axis; if zero, month names will not be abbreviated, and only the months which fit the space available on the axis will be displayed. The default is zero - months are not abbreviated.

!PDT.Start Level

Corresponding Plot Keyword: Start_Level

Allows you to override the starting level of the Date/Time axis labels that PV-WAVE derives. The default is -1, which causes starting levels to be selected automatically.

!PDT.DT_Crange

Contains axis ranges generated by the PLOT procedure.

!PDT.DT_Range

Corresponding Plot Keyword: DT Range

Can be used to specify an exact Date/Time axis range. You must pass in the desired start and end values from a Date/Time Julian value. The specified range may be adjusted slightly depending on the data. To force an exact axis range (exactly as specified), set the XStyle keyword to two. See the description of XStyle in Chapter 3, Graphics and Plotting Keywords.

!PDT.DT Offset

This is a read-only system variable, which is mainly for internal PV-WAVE use.

!PDT.Week_Boundary

Corresponding Plot Keyword: Week_Boundary

Allows you to set a different day of the week as the boundary for numbering the start of the week on the axis levels. (O = Sunday, 1= Monday, etc.)

!Pi

The floating-point value of pi. This is a read-only system variable.

!Product

(Read only) Identifies the product being run: either "cl" or "advantage".

!Prompt

A string variable containing the prompt used by PV-WAVE.

!Quarter Names

Array of strings containing the names for fiscal quarters. The default names of these labels are Q1, Q2, Q3, and Q4.

!Quiet

If set to 1, suppresses compiler messages from being shown on the display screen.

!Radeg

A floating-point value for converting radians to degrees. The value is $180 / \pi$ or approximately 57.2958. This is a read-only system variable.

!Start

Contains the value of the time at which you started PV-WAVE as a Date/Time structure.

!Time Separator

A string containing, by default, a colon (:) character. This character is used to separate the parts of a time (for example, 07:54:58.000). To use a different character, change the value of this variable. This variable is used by the DT_PRINT, DT_TO_STR, and DC_WRITE_* routines.

!Version

A structure whose four string fields contain the current operating system, architecture and name (platform) of the machine running PV-WAVE, and the current PV-WAVE version number. This is a read-only system variable.

!Weekend List

This system variable is created by the CREATE WEEKENDS procedure. It contains an array of long integers, where ones represent weekends and zeros represent weekdays. The values are defined by the CREATE_WEEKENDS routine.

!X

The system variables !X, !Y, and !Z, are structures that affect the appearance and scaling of the three axes. The fields for !X are described here. !Y and !Z have identical fields with identical meanings and usage.

In addition, almost all fields have corresponding keyword parameters, with identical function, but with temporary effect. For example, to suppress the minor tick marks on the X axis using the !X system variable:

```
!X.Minor = -1
```

while to suppress them in the call to PLOT:

```
PLOT, X, Y, XMinor = -1
```

The name of the keyword parameter is simply the name of the system variable field, prefixed with the letter X, Y, or Z.

The fields for the !X system variable are explained in the following sections.

!X.Charsize

Corresponding Plot Keyword: XCharsize

The size of the characters used to annotate the axis and its title. This field is a scale factor applied to the global scale factor. For example, setting !P.Charsize to 2.0 and !X.Charsize to 0.5 results in a character size of 1.0 for the X axis.

!X.Crange

The *output* axis range. Setting this variable has no effect; set !X.Range to change the range. !X.Crange(0) always contains the minimum axis value, and !X.Crange(1) contains the maximum axis value of the last plot.

!X.Gridstyle

Corresponding Plot Keyword: XGridstyle

Lets you change the default linestyle of X axis tick marks. The default is a solid line. Other linestyle choices and their index values are listed in the following table:

Table 4-5: Linestyles

Index	Linestyle	
0	Solid	
1	Dotted	
2	Dashed	
3	Dash Dot	
4	Dash Dot Dot Dot	
5	Long Dashes	

!X.Margin

Corresponding Plot Keyword: XMargin

A two-element array specifying the margin on the left (top) and right (bottom) sides of the plot window, in units of character size. The plot window is the rectangular area that contains the plot data (i.e., the area enclosed by the axes).

The default values for !X.Margin are [10, 3], which yields a 10character-wide left margin and a 3-character-wide right margin. The values for !Y.Margin are [4, 2], for a four-character-high bottom margin and a two-character-high top margin.

When calculating the size and position of the plot window, PV-WAVE first determines the plot region, the area enclosing the window plus the axis annotation and titles. It then subtracts the appropriate margin from each side, obtaining the window.

!X.Minor

Corresponding Plot Keyword: XMinor

The number of minor tick intervals. If !X.Minor is 0, the default, the number of minor ticks is automatically determined. You can force a given number of minor tick intervals by setting this field to the desired number. To suppress minor tick marks, set !X.Minor to -1.

!X.Range

Corresponding Plot Keyword: XRange

The input axis range, a two-element vector. The first element is the axis minimum, and the second is the maximum. Set this field, or use the corresponding keyword parameter, to specify the data range to plot. Because of the axis end point rounding, the final axis range may not be equal to this input range. The field !X.Crange contains the axis range used for the plot.

Set both elements equal to 0 for automatic axis ranges:

```
!X.Range = 0
```

Example

To force the X axis to run from 5.5 to 8.3:

```
!X.Range = [5.5, 8.3]
PLOT, X, Y
```

Alternatively, by using keywords:

```
PLOT, X, Y, XRange = [5.5, 8.3]
```

Note that even though the range was set to (5.5, 8.3), the resulting plot has a range of (5.5, 8.5), because of the axis rounding. To inhibit rounding, set !X.Style to 1.

!X.Region

Corresponding Plot Keyword: None.

Contains the normalized coordinates of the region. This field is similar to !X.Window in that it is set by the graphics procedures and is a two-element floating-point array.

!X.S

The scaling factors for converting between data coordinates and normalized coordinates (a two-element array). The formula for conversion from data (X_d) to normalized (X_n) coordinates is:

$$X_n = S_i X_d + S_0$$

If logarithmic scaling is in effect, substitute $\log_{10} X_d$ for X_d .

!X.Style

Corresponding Plot Keyword: XStyle

The style of the axis encoded as bits in a longword. The axis style may be set to exact, extended, none, or no box using this field. See the following table for details.

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 this bit is set, the axes are extended by 5% in each direction, leaving a border around the data.
2	4	None. If this bit is set, the axis and its annotation is 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 or right axis.

Inhibits setting the Y axis minimum value to zero, when the data are all positive and non-zero. The keyword YNozero sets this bit temporarily.

Example

To set the X axis style to exact, use:

$$!X.Style = 1$$

or by using a keyword parameter:

PLOT,
$$X$$
, Y , $XStyle = 1$

!X.Thick

The thickness of the axis line and tick marks. 1.0 is normal.

!X.Tickformat

Corresponding Plot Keyword: XTickformat

Changes the default format for X axis tick labels using FORTRAN-style format specifiers. For example, the following statement changes the default tick label format to floating-point numbers carried to two decimal places.

```
!X.Tickformat = '(F5.2)'
```

The specified width of the format is at least five characters; however, this width expands automatically to accommodate larger values.

For more information on format specifiers, see Explicitly Formatted Input and Output on page 165 of the PV-WAVE Programmer's Guide. See also Example 3: Specifying Tick Label Formats on page 82 of the PV-WAVE User's Guide.

Note that only the I (integer), F (floating-point), and E (scientific notation) format specifiers can be used with !X.Tickformat. Also, you cannot place a quoted string inside a tick format. For example, ("<", F5.2, ">") is an invalid !X.Tickformat specification.

!X.Ticklen

Corresponding Plot Keyword: XTicklen

The length of tick marks, expressed as a fraction of the plot size. The default value is 0.02. A negative value makes tick marks that extend outside the window, rather than inward.

!X.Tickname

Corresponding Plot Keyword: XTickname

The annotation for each tick. A string array of up to 30 elements. Setting elements of this array allows direct specification of the tick label. If this element contains a null string, the default value, PV-WAVE annotates the tick with its numeric value. Setting the element to a 1-blank string suppresses the tick annotation.

Example

To produce a plot with an abscissa labeled with the days of the week:

```
!X.Tickname = ['SUN', 'MON', 'TUE', 'WED', $
   'THU', 'FRI', 'SAT']
       Set up X axis tick labels.
```

```
!X.Ticks = 6
```

Use six tick intervals, requiring seven tick labels.

```
PLOT, Y
```

Plot the data, this assumes that Y contains 7 elements.

The same plot can be produced, using keyword parameters, with:

```
PLOT, Y, XTickname = ['SUN', 'MON', 'TUE', $
   'WED', 'THU', 'FRI', 'SAT'], XTicks = 6
       Set fields, as above, only temporarily.
```

See Chapter 7, Working with Date/Time Data, in the PV-WAVE User's Guide for information on creating axes with date and time data.

!X.Ticks

Corresponding Plot Keyword: XTicks

The number of major tick intervals to draw for the axis. If !X.Ticks is set to 0, the default, PV-WAVE will select from three to six tick intervals. Setting this field to n, where n > 0, produces exactly n tick intervals, and n + 1 tick marks.

!X.Tickv

Corresponding Plot Keyword: XTickv

The data values for each tick mark, an array of up to 30 elements. You can directly specify the location of each tick by setting !X.Ticks to the number of tick marks (the number of intervals +1) and storing the data values of the tick marks in !X.Tickv. If, by default, !X.Tickv(0) is equal to !X.Tickv(1), PV-WAVE will automatically determine the value of the tick mark.

!X.Title

Corresponding Plot Keyword: XTitle

A string containing the axis title.

!X.Type

Corresponding Plot Keyword: XType

Specifies the type of axis, 0 for linear, 1 for logarithmic, 2 for Date/Time.

!X.Window

Contains the normalized coordinates of the axis end points, the plot window. This field is set by PLOT, CONTOUR, and SURFACE. Changing its value has no effect. It is a two-element floating-point array.

!Y

The main Y axis system variable structure. !Y and !X have identical fields with identical meanings and usage. See !X for more information.

!Z

The main Z axis system variable structure. !Z and !X have identical fields with identical meanings and usage. See !X for more information.

APPENDIX

Software Character Sets

Software Character Sets

```
Font 3, Simplex Roman

A B C D E F F G H I J J

KK L MM NN O PP Q R R S T

U V W X Y Z Z [[ \ \ ] ] ^^

- ' a D b c d d e e f g g h

i j kk | mm nn o pp qq , r

s t u u v w x x y y z Z !! "

## $$ % & .' (( ) ) ** + + ,

- . . // o 0 1 2 2 3 3 4 4 5 5 6 6

7 8 8 9 : : ; < = > ? @
```

Font 4, Simplex Greek A^{A} B^{B} C^{Γ} D^{Δ} E^{E} F^{Z} G^{H} H^{O} I^{Γ} J^{K} K^{Λ} L^{M} M^{N} N^{Ξ} O^{O} P^{Π} Q^{P} R^{Σ} S^{T} T^{Υ} U^{Φ} V^{X} W^{Ψ} X^{Ω} Y^{∞} $Z^{\mathcal{I}}$ I^{Γ} I^{Λ} I^{Λ} I^{A} I^{A}

Font 5, Duplex Roman AA BB c D E F G H I J $_{N}N$ $_{o}O$ $_{P}P$ $_{Q}Q$ $_{R}R$ $_{s}S$ $_{T}T$ ĽΚ , L M_M \sim [$_{1}$ $_{2}$ $_{3}$ $_{4}$ $_{4}$ $_{4}$ $_{4}$ _ .' ad bb cd de f gg k l mm n o $p_p q_q$ z Z !! "'' uu v ww $_{x} \times _{y} y$ ## \$\$ %% && .' ((₎) // 00 11 22 33 44 55 9 ; ; , < = >

```
Font 6, Complex Roman
AA BB CC DD EE FF GG HH I J
K L MM NN OO PP OQ RR SS T
\sim [ \sim ], Z_z Y_y W_w V_y U_u
__ 'aabccddee,fgg
i j k l mm n o p q r
  t uu vv ww xx yy zz !! "''
$$ %% && ' (( )) ** ++ ,
_{-} _{-} _{/} _{0} _{0} _{1} _{2} _{3} _{4} _{5} _{5}
\frac{7}{7} \frac{8}{8} \frac{9}{9} : ; \frac{5}{5} \frac{1}{2} \frac{1}{2} \frac{1}{2} \frac{1}{2}
```

Font 7. Complex Greek $_{\mathbf{A}}\mathbf{A}$ $_{\mathbf{B}}\mathbf{B}$ $_{\mathbf{C}}\mathbf{\Gamma}$ $_{\mathbf{D}}\mathbf{\Delta}$ $_{\mathbf{E}}\mathbf{E}$ $_{\mathbf{F}}\mathbf{Z}$ $_{\mathbf{G}}\mathbf{H}$ $_{\mathbf{H}}\mathbf{\Theta}$ $_{\mathbf{I}}\mathbf{I}$ $_{\mathbf{J}}\mathbf{K}$ N $^{\Xi}$ O P $^{\Pi}$ Q R $^{\Sigma}$ T $^{\tau}$ Λ M $_{\rm s} \tau$ $_{\rm t} v$ $_{\rm u} \phi$ $_{\rm v} \chi$ $_{\rm w} \psi$ $_{\rm x} \omega$ $_{\rm y} \infty$ $_{\rm z} l$ $_{\rm !} !$." ## \$\$ %% && .' (()) ** + - . . // 00 11 22 33 44 55 $_{7}$ $_{8}$ $_{9}$ $_{9}$ $_{1}$ $_{1}$ $_{2}$ $_{3}$ $_{4}$ $_{5}$ $_{7}$ $_{8}$ $_{9}$ $_{9}$ $_{1}$ $_{2}$ $_{3}$ $_{4}$ $_{5}$ $_{7}$ $_{8}$ $_{9}$ $_{9}$ $_{1}$ $_{2}$ $_{3}$ $_{4}$ $_{5}$ $_{7}$ $_{8}$ $_{9}$ $_{9}$ $_{1}$ $_{2}$ $_{3}$ $_{4}$ $_{5}$ $_{7}$ $_{8}$ $_{9}$ $_{9}$ $_{1}$ $_{2}$ $_{3}$ $_{4}$ $_{5}$ $_{7}$ $_{8}$ $_{9}$ $_{9}$ $_{1}$ $_{2}$ $_{3}$ $_{2}$ $_{3}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{3}$ $_{3}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{3}$ $_{3}$ $_{3}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{3}$ $_{3}$ $_{3}$ $_{3}$ $_{4}$ $_{2}$ $_{3}$ $_{3}$ $_{3}$ $_{3}$ $_{3}$ $_{4}$ $_{3}$ $_{4}$ $_{3}$ $_{4}$ Font 8, Complex Italic

```
Font 10, Special \mathbf{z} \mathbf{z}
       # $ % & \cdot \cdo
```

Font 11, Gothic English A B C C D E E F F G H I J J KW LU MM NY OU PH QU RR S T 7 - 89 - 99 : ; < = > ?

Font 13, Complex Script $A^{\mathcal{A}} = B^{\mathcal{B}} = C^{\mathcal{C}} = D^{\mathcal{D}} = E^{\mathcal{C}} = F^{\mathcal{F}} = G^{\mathcal{G}} + H^{\mathcal{H}} = F^{\mathcal{F}} = F^{\mathcal{F}} = G^{\mathcal{G}} + H^{\mathcal{H}} = F^{\mathcal{F}} = F^{\mathcal{F}} = G^{\mathcal{G}} + H^{\mathcal{H}} = F^{\mathcal{F}} = F^{\mathcal{F}} = G^{\mathcal{G}} = H^{\mathcal{H}} = F^{\mathcal{F}} = F^{\mathcal{F}} = G^{\mathcal{G}} = H^{\mathcal{H}} = F^{\mathcal{F}} = F^{\mathcal{F}} = F^{\mathcal{F}} = G^{\mathcal{G}} = H^{\mathcal{H}} = F^{\mathcal{F}} = F^{\mathcal{F$

Font 14. Gothic Italian

Font 15. Gothic German

Font 16, Cyrillic

AA BB cB DF EД FE GЖ H3 I И JЙ

KК LЛ MM NH OO PП QP RC sT TУ

UФ VX WЦ хЧ УШ ZЩ [Ы (Э] Ь .Ю

Я .' aa b б cB d Г еД f е g ж h 3

I И ј Й к К I Л m M n H OO PП q Р C

s Т t У uф v X wц хЧ уш ZЩ ! ... Ю

Font 17, Triplex Roman

AA BB cC DD EE FF GG HH I J J

KK LL MM NN OO PP QQ RR SS T

UU VV WW XX YY ZZ [{ \ \ \]} .^

' a b c d d ee f gg h

i j kk l mm nn oo pp qq r

s t u v v ww x x yy z z !! "

\$\$ %% && ' (()) .* + + ,

- . . / 00 11 22 33 44 55 66

7 88 99 : ; ; < = , > ? @@

Font 18, Triplex Italic

Font 20, Miscellaneous

A B C D E F G H I I J J
$$\stackrel{\leftarrow}{}$$

K T L M N N O R Q Q R S T $\stackrel{\leftarrow}{}$

U V V W X Y Z $\stackrel{\leftarrow}{}$

I D M N N O R Q Q R $\stackrel{\leftarrow}{}$

I D M N N O R Q Q R $\stackrel{\leftarrow}{}$

I D M N N O R Q Q R $\stackrel{\leftarrow}{}$

I D M N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

I D N N O R Q Q R $\stackrel{\leftarrow}{}$

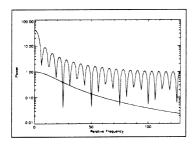
I D N N O R Q Q R $\stackrel{\leftarrow}{}$

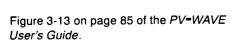
I D N N O R



Picture Index

2D Plots





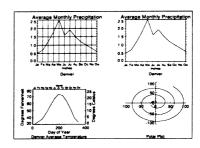
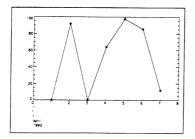


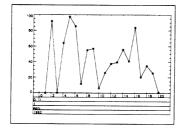
Figure 3-12 on page 84 of the *PV=WAVE* User's Guide.



| 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |

Figure 7-2 on page 245 of the *PV=WAVE* User's Guide.

Figure 7-1 on page 222 of the *PV=WAVE* User's Guide.



SAT Scores, 1967

510

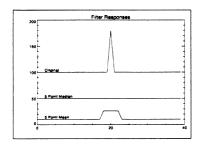
500

400

Fames Vettel Mess Vetter Fames Nam Mass Nam

Figure 7-3 on page 246 of the *PV=WAVE* User's Guide.

Figure 3-10 on page 80 of the *PV=WAVE User's Guide*.



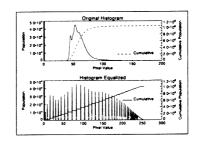
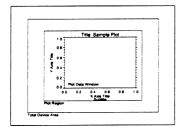


Figure 5-2 on page 160 of the *PV=WAVE* User's Guide.

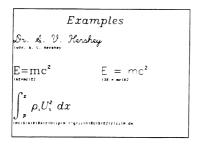
Figure 5-1 on page 157 of the *PV=WAVE User's Guide*.



2000

Figure 3-14 on page 86 of the *PV=WAVE* User's Guide.

Figure 7-8 on page 253 of the *PV=WAVE* User's Guide.



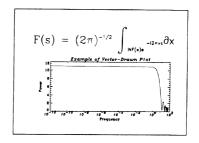
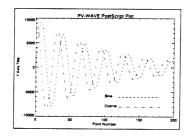
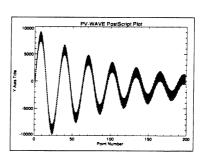


Figure 9-3 on page 300 of the *PV*=*WAVE User's Guide*.

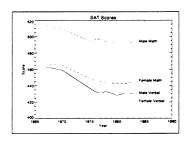
Figure 9-4 on page 303 of the *PV=WAVE* User's Guide.





See page A-42 of the PV-WAVE User's Guide.

See page A-47 of the *PV-WAVE User's* Guide.



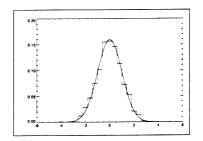
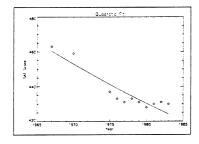


Figure 3-4 on page 67 of the *PV=WAVE* User's Guide.

Figure 3-6 on page 71 of the *PV=WAVE User's Guide*.



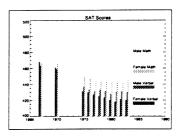
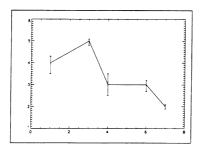


Figure 3-7 on page 73 of the *PV=WAVE* User's Guide.

Figure 3-9 on page 77 of the *PV=WAVE* User's Guide.



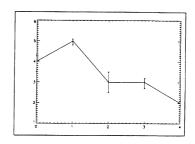


Figure 2-14 on page 308 of the *PV=WAVE* Reference, Volume 1.

Figure 2-13 on page 307 of the *PV=WAVE* Reference, Volume 1.

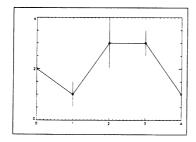


Figure 2-29 on page 506 of the *PV*=*WAVE* Reference, Volume 1.

Figure 2-41 on page 586 of the *PV*=*WAVE* Reference, Volume 1.

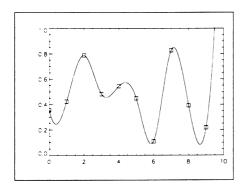


Figure 2-23 on page 490 of the *PV=WAVE* Reference, Volume 1.

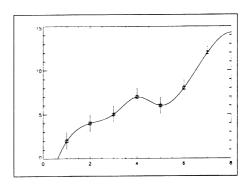


Figure 2-25 on page 493 of the *PV=WAVE* Reference, Volume 1.

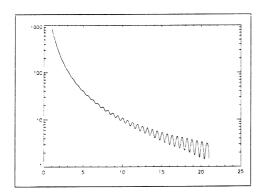


Figure 2-27 on page 501 of the *PV*=*WAVE User's Guide*.

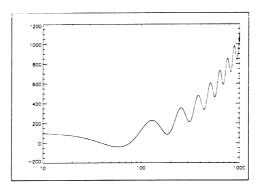


Figure 2-28 on page 502 of the *PV=WAVE* User's Guide.

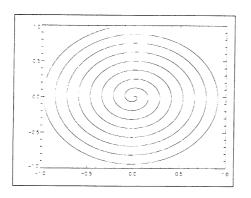


Figure 2-34 on page 513 of the *PV=WAVE* Reference, Volume 1.

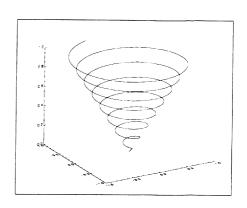


Figure 2-35 on page 514 of the *PV=WAVE* Reference, Volume 1.

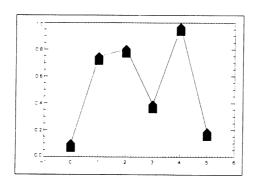


Figure 2-59 on page 237 of the *PV*=*WAVE* Reference, Volume 2.

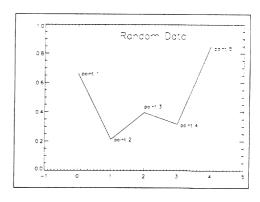


Figure 2-83 on page 492 of the *PV*=*WAVE* Reference, Volume 2.

Contour Plots

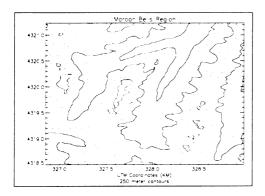


Figure 4-2 on page 100 of the *PV*=*WAVE User's Guide*.

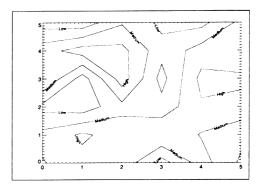


Figure 4-7 on page 108 of the *PV=WAVE* User's Guide.

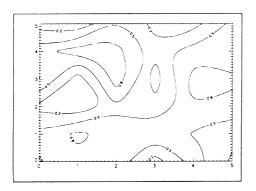


Figure 4-8 on page 109 of the *PV=WAVE* User's Guide.

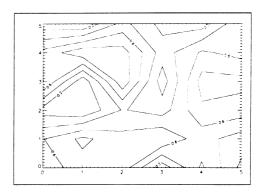


Figure 4-5 on page 106 of the *PV*=*WAVE User's Guide*.

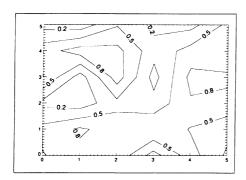


Figure 4-6 on page 107 of the *PV*=*WAVE User's Guide*.

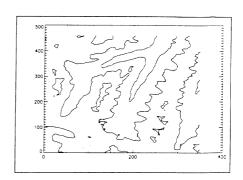


Figure 4-1 on page 97 of the *PV*=*WAVE User's Guide*.

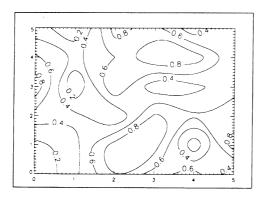


Figure 2-7 on page 122 of the *PV=WAVE* Reference, Volume 1.

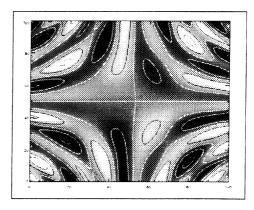


Figure 2-21 on page 404 of the *PV=WAVE* Reference, Volume 1.

3D Plots, Surface and Shaded Surface Plots

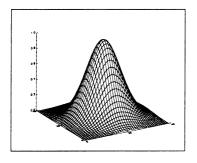


Figure 4-10 on page 113 of the *PV*=*WAVE* User's Guide.

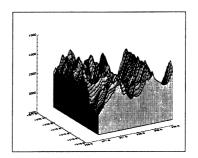


Figure 4-11 on page 114 of the *PV*=*WAVE* User's Guide.

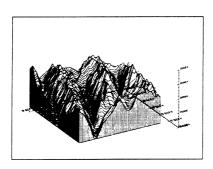


Figure 4-11 on page 114 of the *PV=WAVE* User's Guide.

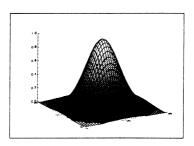
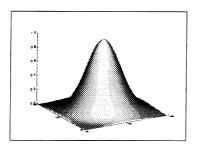


Figure 4-16 on page 133 of the *PV=WAVE* User's Guide.

PV-WAVE Picture Index A-11



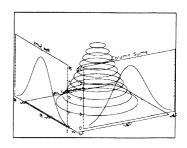
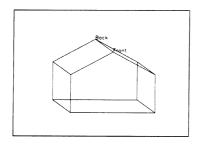


Figure 4-16 on page 133 of the *PV*=*WAVE User's Guide*.

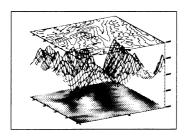
Figure 4-14 on page 127 of the *PV*=*WAVE User's Guide*.



20×10⁵
15×10⁵
50×10⁶

Figure 4-12 on page 124 of the *PV=WAVE* User's Guide.

Figure 5-4 on page 166 of the *PV=WAVE* User's Guide.



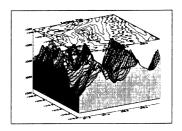


Figure 4-15 on page 129 of the *PV=WAVE* User's Guide.

Figure 4-13 on page 126 of the *PV=WAVE* User's Guide.

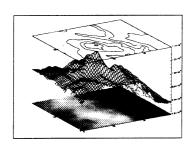


Figure 2-53 on page 106 of the *PV=WAVE* Reference, Volume 2.

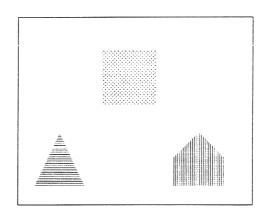


Figure 2-38 on page 545 of the *PV=WAVE* Reference, Volume 1.

PV-WAVE Picture Index A-13

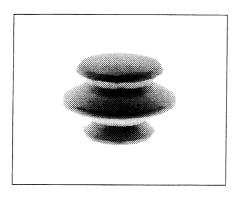


Figure 2-40 on page 567 of the *PV*=*WAVE* Reference, Volume 1.

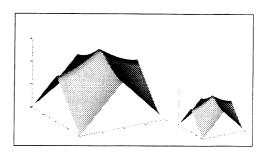


Figure 2-42 on page 23 of the *PV*=*WAVE* Reference, Volume 2.

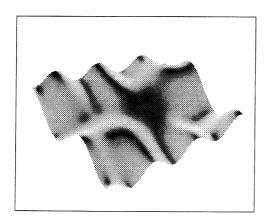


Figure 2-50 on page 95 of the *PV=WAVE* Reference, Volume 2.

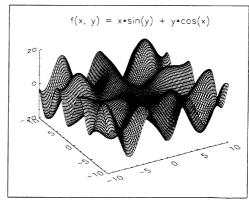


Figure 2-56 on page 171 of the *PV*=*WAVE* Reference, Volume 2.

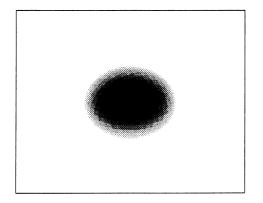


Figure 2-46 on page 72 of the *PV=WAVE* Reference, Volume 2.

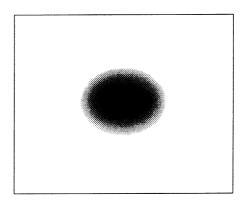


Figure 2-47 on page 73 of the *PV=WAVE* Reference, Volume 2.

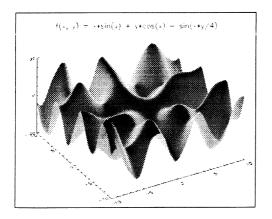


Figure 2-48 on page 87 of the *PV=WAVE* Reference, Volume 2.

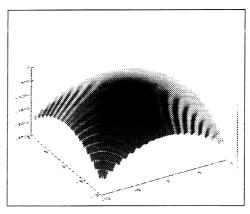
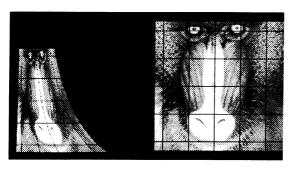


Figure 2-49 on page 90 of the *PV=WAVE* Reference, Volume 2.

Images



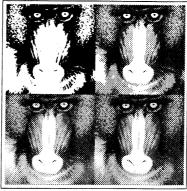


Figure 5-6 on page 172 of the *PV=WAVE* User's Guide.

See page A-43 of the *PV-WAVE User's* Guide.



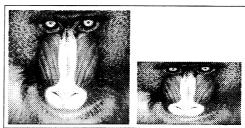


Figure 2-8 on page 128 of the *PV=WAVE Reference*, *Volume 1*.

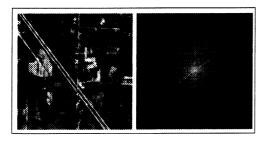
Figure 2-6 on page 117 of the *PV=WAVE* Reference, Volume 1.





Figure 2-12 on page 303 of the *PV=WAVE*Reference, Volume 1.

Figure 2-11 on page 258 of the *PV=WAVE* Reference, Volume 1.



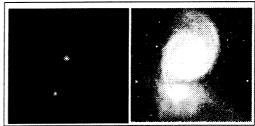
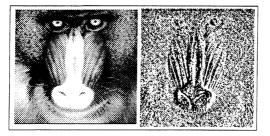


Figure 2-15 on page 330 of the *PV=WAVE*Reference, Volume 1.

Figure 2-16 on page 385 of the *PV=WAVE* Reference, Volume 1.

PV-WAVE Picture Index A-17



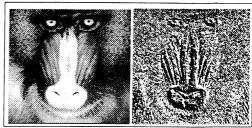


Figure 2-51 on page 102 of the *PV*=*WAVE* Reference, Volume 2.

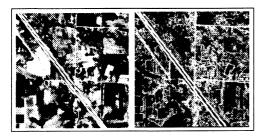
Figure 2-52 on page 103 of the *PV*=*WAVE* Reference, Volume 2.





Figure 2-43 on page 47 of the *PV*=*WAVE* Reference, Volume 2.

Figure 2-54 on page 121 of the *PV=WAVE* Reference, Volume 2.



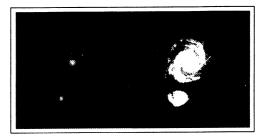


Figure 2-55 on page 124 of the *PV=WAVE* Reference, Volume 2.

Figure 2-2 on page 76 of the *PV=WAVE*Reference, Volume 1.

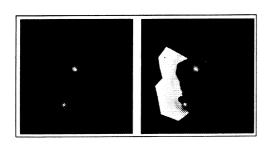


Figure 2-9 on page 234 of the *PV=WAVE* Reference, Volume 1.

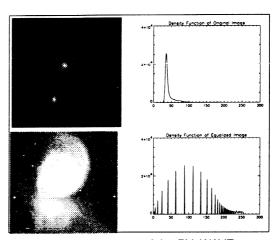


Figure 2-17 on page 392 of the *PV=WAVE* Reference, Volume 1.

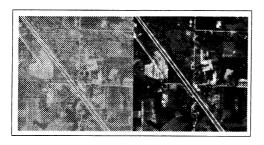


Figure 2-58 on page 229 of the *PV=WAVE* Reference, Volume 2.

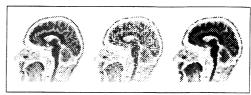


Figure 2-22 on page 456 of the *PV=WAVE* Reference, Volume 1.

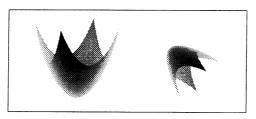


Figure 2-44 on page 50 of the *PV=WAVE Reference*, *Volume 2*.

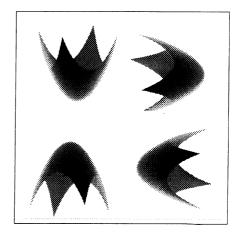


Figure 2-45 on page 53 of the *PV=WAVE* Reference, Volume 2.

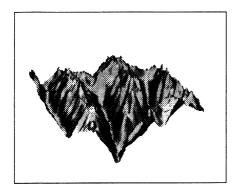


Figure 4-17 on page 134 of the *PV=WAVE* User's Guide.

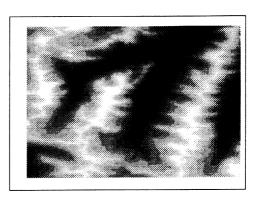


Figure 4-3 on page 101 of the *PV=WAVE User's Guide*.

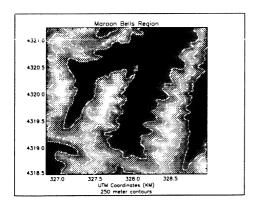


Figure 4-4 on page 103 of the *PV=WAVE* User's Guide.

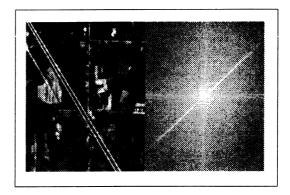


Figure 5-3 on page 166 of the *PV=WAVE* User's Guide.

Rendered Images

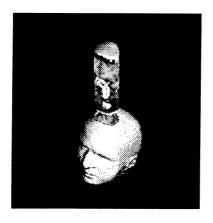


Figure 6-8 on page 216 of the *PV=WAVE* User's Guide.

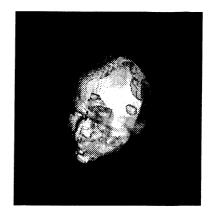


Figure 6-7 on page 214 of the *PV=WAVE* User's Guide.

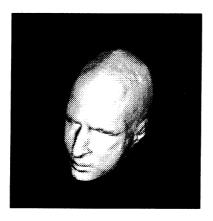


Figure 6-4 on page 205 of the *PV=WAVE* User's Guide.

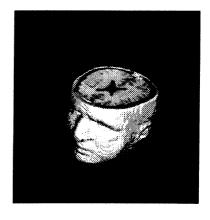


Figure 6-6 on page 212 of the *PV*=*WAVE* User's Guide.



Figure 6-5 on page 210 of the *PV*=*WAVE User's Guide*.

PV=WAVE Picture Index A-23

Multivolume Index

access mode, for VMS PG-225

This is a multivolume index that includes references to the *PV-WAVE User's Guide* (UG), *PV-WAVE Programmer's Guide* (PG), and both volumes of the *PV-WAVE Reference* (R1 and R2).

accessing data in PV-WAVE PG-350 Symbols accumulated math error status R1-90, PG-263 ! character UG-34 ACOS function R1-31 # operator PG-46 actual parameters PG-234, PG-236 \$ character UG-34 ADD_EXEC_ON_SELECT procedure & character UG-35 R1-33 'character UG-34 addition operator (+) " character UG-35 examples PG-45 () operator PG-44 for concatenating strings PG-123 * character UG-36 operator precedence PG-32 * operator PG-45, PG-85 ADDSYSVAR procedure R1-34 *, in array notation PG-85 ADJCT procedure R1-36, UG-331 + operator PG-45, PG-123 allocating colors UG-312 - operator PG-45 ALOG function R1-38 . character UG-35 ALOG10 function R1-40 / operator PG-45 ambient component of color UG-199 : character UG-36 ampersand ; character UG-34 in command files UG-21 after @ symbol UG-20 separating multiple statements < operator PG-46 UG-35 = operator PG-44 AND operator > operator PG-46 description of PG-48 @ character UG-36 operator precedence PG-33 ^ operator PG-46 truth table PG-43 animation controlling the pace R2-280, R2-315 A format code PG-A-9, PG-A-11 cycling through images R2-280 aborting double-buffering on 24-bit displays plots UG-62 UG-A-79 PV-WAVE UG-15-UG-16 of data in pixmaps R2-278, R2-316 ABS function R1-30, PG-21 absolute value R1-30, PG-21

PV-WAVE Index i

of images R2-277, R2-315	arithmetic operations, overflow condi-
of images using pixmaps UG-A-85	tion PG-19
annotation	arrays
alignment of the text R2-497	* in notation PG-85
axes tick marks R2-527, R2-531,	accessing
R2-535, R2-559	by memory order PG-281,
centering of text R2-497	PG-282
character size R2-501, R2-518,	large arrays PG-281
R2-543, R2-554	arbitrary type, creating R1-449
color of R2-504	assignment statements PG-55-
contour plots R2-499	PG-56
controlling tick marks of UG-78	average of R1-46
display without data R2-511	building tables from R1-63
margin for R2-525, R2-530, R2-534	calculating determinant of R1-248
on axes R2-524, R2-529, R2-533	columns in PG-47
plots R2-490, UG-68-UG-69	combining PG-88-PG-91
positioning text, example UG-91	concatenation operators PG-8,
three-dimensional orientation of	PG-47
text R2-521	contouring 2D UG-94
width of text string R2-524	correlation coefficient of R1-133
with hardware fonts UG-291	creating
with software fonts UG-294,	dynamically R1-449
UG-302	from unique elements R2-230
X axis R2-520, R2-527	with arbitrary initialization
Y axis R2-531	R2-31
Z axis R2-535	day of the year for each date in a
application programming	date/time variable R1-155
create special effects with	definition of PG-4
graphics functions UG-A-94	degenerate dimensions PG-86
the write mask UG-A-94	double-precision type R1-156
look-and-feel of GUI PG-411,	elements, number of R1-469
UG-51	eliminating unnecessary dimen-
not enough colors available	sions PG-83
UG-A-81	extracting fields from PG-37
providing a GUI UG-A-70	finding the maximum value R1-452
setting resources PG-465	floating type R1-340, R1-343
unexpected colors UG-A-96	functions to create from various data
WAVE Widgets PG-412, PG-414	types PG-39
Widget Toolbox PG-479, PG-494	in structures PG-112, PG-114
arc-cosine R1-31	indexed by rows and columns
arcsine R1-41	PG-81
arctangent R1-45	integer type, creating R1-409,
area of polygon R1-522	R1-413
arithmetic errors, checking for PG-263,	linear log scaling R1-497
PG-265	locating non-zero elements in R2-354

ii PV-WAVE Index

log-linear scaling R1-497	ASCII
log-log scaling R1-497	files
longword type, creating R1-442	formatted PG-144
masking PG-49	reading R1-161, R1-177
mean of R2-136	row-oriented PG-146
median value of R1-454	fixed format PG-155, PG-157
minimum value of R1-462	free format PG-155-PG-156
number of elements R1-469	free format data, writing to a file
reading	R1-203, R1-212
data into PG-180, PG-181	input/output
into from display R2-223,	comparison with binary
UG-142	PG-151
with READF PG-181	procedures PG-155
referencing a non-existent element	ASIN function R1-41
PG-83	assignment operator (=) PG-44
reformatting R2-24	assignment statements
replicating values of R2-31	description of PG-9
resizing of R1-125, R2-21	examples PG-53-PG-54
rotating R2-48	form 4 PG-56
rows in PG-47	four forms of PG-53
scaling to byte values R1-73	using array subscripts with
setting default range of R2-81	form 2 PG-55
shifting elements of R2-99	form 4 PG-57
size and type of R2-114	with associated variables PG-59
smoothing of R2-119	ASSOC function R1-43
sorting contents of R2-125	description PG-154
standard deviation of R2-136	example of PG-213, PG-214
storing elements with array sub-	exporting image data with PG-191
scripts PG-91	general form PG-212
string array with names of days of the	in relation to UNIX FORTRAN pro-
month R1-465	grams PG-218
string array with names of days of the	offset parameter PG-212, PG-216
week R1-153	use of PG-152
string, creating R2-111, R2-139	associated variables R1-43
subarrays PG-86	See also ASSOC function
subscripts	description PG-24
of other arrays PG-87-PG-88	in assignment statements PG-59
of points inside polygon region	asterisk, for subscript ranges UG-36
R1-542, R1-550	at (@) character UG-36
storing elements PG-91	ATAN function R1-45
subsetting PG-42	attachments of widgets PG-422—
sum elements of R2-200	PG-424
transposition of R2-204	attributes
transposition of rows and columns	See also resource file
PG-47	of an expression PG-272

PV-WAVE Index iii

of variables PG-24	specification R2-526-R2-534,
VMS record PG-226-PG-227	UG-65
average, boxcar R2-119	specifying data to plot R2-556
AVG function R1-46	saving the scaling parameters
axes	R2-517
See also AXIS procedure	scaling of R2-557, UG-63, UG-84
See also tick marks	setting
adding to plots UG-87	style R2-557
additional R1-48	thickness R2-558
annotation of tick marks R2-527,	size of annotation R2-524, R2-529,
R2-531, R2-535, R2-559	R2-533
box style R2-526, R2-531, R2-534	styles of UG-64
color of R2-504	suppressing of R2-526
controlling	title of R2-529, R2-532, R2-535,
appearance R2-554	R2-560
scaling R2-554	AXIS procedure R1-48, UG-87-UG-89
converting between data and nor-	
malized coordinates R2-557	В
coordinate systems for UG-59,	
UG-87, UG-89	background color
data range of R2-526, R2-530,	LJ-250 output UG-A-23
R2-534	PCL output UG-A-27
data values of tick marks R2-560	plotting background color R2-498
date/time UG-221, UG-245	PostScript output UG-A-40
drawing and annotating UG-87	setting R2-543
drawn without data R2-511	SIXEL output UG-A-61
endpoints R2-560	backing store, for windows UG-A-7
exchange of UG-126	back-substitution for linear equations
inhibiting drawing of UG-88	R2-177
intervals R2-525	bandpass filters R1-250, UG-164
length of tick marks R2-522—	bar charts UG-76-UG-79
R2-535, R2-559	bar, menu PG-426
linear R2-560	base 10 logarithm R1-40
logarithmic R2-529, R2-532,	basis function, example of R1-137
R2-536	batch files
scaling R2-560, UG-84	See command files
multiple R1-48, UG-68, UG-87	BEGIN statement PG-60
output axis range R2-554	BESELI function R1-51
outward pointing tick marks R2-522	BESELJ function R1-53
positioning of UG-85	BESELY function R1-55
range	BILINEAR function R1-57
input range R2-556	bilinear interpolation R1-57, UG-141,
max and min of last plot	UG-170
R2-554	

iv PV-WAVE Index

binary	operator precedence PG-32
data PG-152	OR PG-43, PG-50
advantages and disadvantages	table of PG-7, PG-43
of PG-151	XOR PG-50
input/output procedures	Bourne shell PG-293, PG-299
PG-188	box style axes R2-526, R2-531, R2-534
input/output, comparison with	BREAKPOINT procedure R1-61
ASCII PG-151	buffer, flushing output R1-294, R1-344
reading FORTRAN files	BUILD_TABLE function R1-63, UG-267
PG-224	bulletin board widget PG-421
transferring PG-154	Butterworth filters UG-163-UG-164
transferring strings PG-197	button box widget R2-406, PG-433
UNIX vs. FORTRAN PG-199	buttons (widgets)
files	active PG-429
comparison to human-readable	iconic PG-430, PG-433
files PG-151	in dialogs PG-454
efficiency of PG-188	in messages PG-451
record length PG-148	menu toggle PG-430
record-oriented PG-149	radio PG-436
VMS PG-149	sensitivity of PG-470
input/output	toolbox PG-433
advantages and disadvantages	BYTARR function R1-66
of PG-151	byte
procedures PG-154	See also BYTE function
routines PG-154	arrays, creating R1-66
transfer of string variables PG-197	data
BINDGEN function R1-60	a basic data type PG-3, PG-17
bit shifting operation R1-418	converting to characters
bitmap, used for cursor pattern UG-A-72	PG-37, PG-125
blank, removing from strings R2-140	reading from XDR files
block mode file access (VMS) PG-225	PG-206-PG-207
blocking window	scaling R1-73
dialog box PG-453	shown in figure PG-146
file selection box PG-457	type
popup message PG-449	converting to R1-68
blocks of statements	extracting from R1-68
BEGIN and END identifiers PG-60	BYTE function
definition PG-10	description R1-68
description of PG-60	for type conversion PG-37
END identifier PG-61	using with STRING function
with IF statements PG-72	PG-199
Boolean operators	with single string argument PG-126
AND PG-43, PG-48	BYTEORDER procedure R1-71
applying to integers PG-43	BYTSCL function R1-73, UG-154
applying to non-integers PG-43	,
NOT PG-43, PG-50	

PV-WAVE Index

j.	writing to PV-WAVE PG-194
C format strings	C shell PG-299
conversion characters PG-A-23	C_EDIT procedure R1-81, UG-331
for data export PG-A-24	CALL_UNIX function R1-77, PG-359—
for data import PG-A-22	PG-381
when to use PG-167	CALL WAVE PG-371-PG-382
C functions	callback
	adding to a Widget Toolbox widget
CALL_WAVE PG-373	PG-485
loading a UT_VAR PG-387, PG-392	definition of PG-413, PG-485
	for menu button PG-428
used with CALL_UNIX PG-364	list PG-485
w_cmpnd_reply PG-368	parameters, Motif PG-C-3-PG-C-6
w_get_par PG-366	parameters, Motil PG-0-3—PG-0-6 parameters, OLIT PG-C-6—
w_listen PG-365	PG-C-12
w_send_reply PG-367	procedures R1-311
w_smpl_reply PG-367	registering R1-309
wavevars PG-321, PG-335	reason PG-485
C programs	scrolling list PG-447
accessing	calling PV-WAVE
from PV-WAVE with SPAWN	cwavec routine PG-333, PG-340
PG-312	cwavefor routine PG-333, PG-340
PV-WAVE's variable data space from PG-350–PG-351	from a C program R1-429, PG-316
as a client PG-382	from a FORTRAN program
as a server PG-380	(wavecmd) PG-318
CALL_WAVE as a client PG-382	from a statically linked program
calling	PG-333
from within PV-WAVE PG-310	from an external program PG-313
PV-WAVE (cwavec) PG-337	in a UNIX unidirectional environ-
PV-WAVE (wavec) PG-337	ments PG-313
using LINKNLOAD function	callwave object module PG-313
R1-429	carriage control
communicating with R2-232	VMS FORTRAN, table of PG-227
creating XDR files PG-207—PG-208	VMS output PG-226
ending PV-WAVE with waveterm	case folding PG-127
command PG-315	case selector expression PG-62
linking to PV-WAVE PG-316	CASE statement PG-10, PG-62
under UNIX PG-347	CD procedure R1-79, PG-302
under VMS PG-349	CeditTool procedure R2-294
reading files with PG-196	CENTER_VIEW procedure R1-86,
running PV-WAVE from PG-335,	UG-194
PG-342	CGM output UG-A-9-UG-A-12
sending commands to PV-WAVE	changing the PV-WAVE prompt R2-552
PG-315	

Vi PV-WAVE Index

characters	child processes PG-309-PG-310
changing case of R2-150, R2-166,	spawning R2-126
PG-127	CINDGEN function R1-93
concatenation PG-123	clear output buffer R1-294, R1-344
converting	clear screen of graphics device R1-298
from byte data PG-125	client
parameters to R2-144, PG-124	C program as PG-382
extracting R2-154, PG-131	definition of PG-359
	in X Window Systems UG-A-69
inserting R2-157, PG-131	linking with PV-WAVE PG-361
locating substring R2-154, PG-131	PV-WAVE as (CALL_UNIX)
non-printable, specifying PG-22	PG-362, PG-377-PG-381,
non-string arguments to string rou-	PG-382, PG-377—PG-301,
tines PG-133	
obtaining length of strings R2-148,	clipping
PG-130	graphics output R2-503
removing white space R2-140,	of graphics window R2-543
R2-162, PG-128	rectangle R2-503
semicolon UG-34	strings R2-158
setting to lowercase R2-150	suppressing R2-511, R2-546
setting to uppercase R2-166	CLOSE procedure R1-94
size of R2-501, R2-518, R2-543	closing
axis notation R2-501, R2-518,	files R1-94
R2-524, R2-529, R2-533,	DC (Data Connection) functions
R2-554	PG-140
software UG-291	description PG-140
special UG-33	LUNs PG-140
string operations supported PG-121	graphics output file UG-A-5
thickness of R2-543	widget hierarchy PG-468, PG-471,
unprintable PG-125	PG-485
vector-drawn UG-291	colon (:)
working with PG-121	format code PG-A-10, PG-A-12
CHEBYSHEV function R1-89	use in PV-WAVE UG-35
CHECK_MATH function R1-90, PG-36,	color
PG-263, PG-265	See also color tables, colormaps
checking for	See also TEK_COLOR procedure
keywords PG-269	as 6-digit hexadecimal value
number of elements PG-270	UG-A-91
overflow in integer conversions	can be viewed in bar R2-284,
PG-264	UG-317, UG-331
parameters in procedures and func-	characteristics, determining
tions PG-269	UG-A-82
positional parameters PG-269	contours, filling UG-109
presence of keywords PG-271	decomposed into red, green, and
size and type of parameters	blue UG-A-79
PG-272	
validity of operands PG-264	
, ,	

PV-WAVE Index VII

editing interactively R2-290, R2-292, R2-301, UG-316,	setting background R2-498,
UG-331	R2-543
histogram equalizing R1-383	setting in WAVE Widgets PG-463
images PG-189	specifying by name UG-A-92
in resource file PG-415	translation
in TIFF files PG-192	See also color table
indices	in X Windows UG-A-84
	table UG-308, UG-A-71,
changing default UG-57	UG-A-78, UG-A-85
definition of UG-57, UG-305 displaying UG-332	true-color UG-147
for Tektronix 4510 rasterizer	unexpected UG-A-96
	vector graphics UG-311, UG-A-92
UG-A-62	with monochrome devices UG-310
interpretation of UG-58	X Windows UG-A-77
list of PG-463	color bar
lists, converting R1-535	purpose of R2-284, UG-317,
making color table with HLS or HSV	UG-331
system UG-331	shown in figure R2-298
map	COLOR field of !P, changing of UG-A-66
See colormap, color tables	color systems
model	definition UG-305
See color systems	HLS UG-309
number available on graphics device	HSV UG-309
R2-538, UG-322	RGB UG-306
obtaining tables from common block	color tables
UG-321	adding new UG-330
of	based on
annotation R2-504	HLS system UG-329-UG-330
axes R2-504	HSV system UG-329
contour levels R2-500	changing predefined UG-330
data R2-504	controlling contrast UG-331
elements inside plots UG-311,	copying R2-67, UG-328
UG-321	creating R1-81, R1-95, R1-97,
graphics output R2-504	UG-149, UG-329
surface bottom R2-499	creating interactively R1-81, R1-97,
on 24-bit display UG-A-79	R1-396, R1-398, R1-400,
on 8-bit display UG-A-79	R1-494, R1-588
PostScript devices UG-A-39	custom, creating R2-293
oseudo PG-189	discussion of UG-145, UG-310
aster graphics UG-A-92	editing UG-149
represented by shades of gray	from Standard Library proce-
PG-189	dures UG-329
eserving for other applications	interactively R2-290, R2-292,
UG-A-84	R2-301, UG-316, UG-331
otate R2-286, R2-296, R2-303	expanding R2-142
unning out of UG-A-81	

viii PV-WAVE Index

histogram equalizing R1-383,	column-oriented
R1-386, UG-329	ASCII files PG-144
indices R2-294	data
listed in table UG-312	in arrays PG-47
loading R2-219, UG-311, UG-329	reading R1-168, R1-184,
from colors.tbl file UG-145	R1-207, R1-216
from variables UG-145,	transposing with rows PG-47
UG-311, UG-330	FORTRAN write PG-181
into device UG-145	combining
predefined tables R1-435	image and contour plots UG-100
lookup table UG-308, UG-A-79	images with three-dimensional
modifying R1-36, R1-464, UG-330	graphics UG-129-UG-130
to enhance contrast UG-155	surface and contour plots UG-125
obtaining UG-321	command files
predefined UG-145, UG-311,	description of UG-19
UG-329	difference from main programs
replacing R1-464	UG-25
reversing R2-302, UG-320	examples UG-20
rotating R2-302, UG-320	executing UG-19
saving in TIFF file PG-192	at startup UG-13, UG-48
stretching R2-142, R2-302,	guidelines for creating UG-21
R2-303, UG-320, UG-330	terminating execution of UG-20
supplied with PV-WAVE R1-435,	three methods of running UG-19
R1-464, UG-145, UG-311,	using ampersand and dollar sign with
UG-329	UG-21
switching between devices UG-327	command interpreter, VMS PG-299
Tektronix 4115 color table R2-195	command line
utility widgets, shown in figure	editing PG-401
R2-298	entering interactive commands
color wheel, shown in figure R2-298	UG-17
COLOR CONVERT procedure R1-95	keys UG-33
COLOR EDIT procedure R1-97,	command recall
UG-331	description of UG-32
COLOR PALETTE procedure R1-101,	using function keys UG-33
UG-332	with INFO procedure PG-401
colormap	command widget R2-410, PG-459
See also color tables	comment field
compared to color table UG-A-78	in statements PG-52
how PV-WAVE	semicolon UG-34
chooses one UG-A-80	common block
obtains UG-A-83	colors UG-321
private, advantages,	COMMON block procedure
disadvantages UG-A-82	examples PG-249
shared, advantages,	COMMON statement PG-63
disadvantages UG-A-81	creating common variables for
uisauvainages Od-A-01	procedures PG-249
	L

PV-WAVE Index ix

definition PG-12	concatenating
not needed with user data PG-467	arrays PG-47
use of variables PG-63	characters PG-123
used to pass client data PG-486-	strings PG-122-PG-123
PG-489	text PG-123
communicating with	concurrent processes R1-309
a child process PG-309	CONE function R1-113, UG-196
operating systems from PV-WAVE	conformance levels, for TIFF PG-192
PG-291	CONGRID function
compilation	description of R1-115
automatic PG-69	using to magnify or minify an image
of one or more statements R1-313	UG-141
run-time PG-273	CONJ function R1-118
COMPILE procedure R1-104	connecting data points with lines UG-70
compiler messages, suppressing	constant shading UG-131
R2-553	constants
compiling	complex PG-20
.RUN with filename PG-239	double-precision PG-20
automatic PG-239, UG-22	examples PG-19
compiler messages, suppressing	floating-point PG-20
PG-29	hexadecimal PG-18
external programs PG-313	integer PG-18
procedures and functions PG-239	numeric PG-4
saving compiled procedure R1-104	octal PG-18
system limits PG-241	ranges of PG-19
under UNIX for LINKNLOAD	string PG-4, PG-21
PG-322, PG-326, PG-328	using correct data types PG-280
with EXECUTE function PG-273	.CONTINUE command UG-16
complex	continuing program execution UG-15
arrays, creating R1-109, R1-112	contour plots
conjugate R1-118	2D arrays UG-94
constants PG-20	adding a Z axis R2-533
data type R1-109, PG-3, PG-18	algorithms used to draw R2-506,
converting to R1-109	UG-95
shown in figure PG-146	annotation of R2-499
numbers PG-21	cell drawing method R2-506
type arrays, creating R1-109	closing open contours with arrays
variables, importing data into	UG-110
PG-160	color of contour levels R2-500
COMPLEX function R1-109, PG-37	coloring of UG-109
COMPLEXARR function R1-112	combining with
compressing date/time variables	PLOT procedure UG-124
R1-272	SURFACE procedure UG-125
compression, of TIFF files PG-192	combining with surface plots and im-
Computer Graphics Metafile	ages R2-104
See CGM	creating R1-119

X PV-WAVE Index

cubic spline interpolation of R2-519	CONV_FROM_RECT function RT-123,
default number of levels R2-511	UG-193
enhancing UG-97	CONV TO RECT function R1-129,
examples UG-94, UG-96-UG-99,	UG-193
UG-102, UG-105	convenience widgets (Motif) PG-B-3
filling with color R1-529, UG-109	conversion characters
filling with pattern R2-513	C code PG-A-23-PG-A-25
follow method algorithm UG-95	C export PG-A-25
gridding irregular data R1-362	C import PG-A-23
labeling R2-501, UG-105	FORTRAN PG-A-8
levels to contour R2-508, R2-511	conversion functions PG-36
line thickness of R2-504	converting
line-following drawing method	between graphics coordinate sys-
R2-506	tems R1-21, R1-123, R1-129,
linestyle of R2-502	R1-538, R1-559, UG-60, UG-193
maximum value to contour R2-510	color lists R1-524, R1-535
overlaying	data to
an image UG-100	See also extracting data
with scalable pixel devices	byte type R1-68, R1-73
UG-102	complex type R1-109
placement of Z axis R2-533	date/time R1-421, R2-58,
shading of R1-529	R2-159, R2-238, UG-223,
size of annotation R2-500	UG-229
smoothing UG-108	double-precision type R1-267
superimposing on surface plots	floating-point type R1-340
R2-517	integer type R1-338
CONTOUR procedure R1-119	longword integer type R1-442
contrast	string type R2-144
control UG-331	strings PG-124
enhancement UG-153, UG-330	date/time variables to
control points, using to correct linear dis-	double-precision variables
tortion UG-170	R1-286
Control- aborting PV-WAVE UG-15	numerical data R1-292
Control-C	string data R1-289
interrupting PV-WAVE UG-15	strings for tables UG-282
using to abort plots UG-62	degrees to radians PG-27
Control-D, exiting PV-WAVE on a VMS	DT TO SEC function UG-258
system UG-14	DT TO STR procedure UG-256
controls box, see sliders	DT TO VAR procedure UG-257
Control-Y UG-16, UG-34	JUL TO DT function UG-233
Control-Z	Julian day number to PV-WAVE
exiting	date/time R1-421
PV-WAVE on a VMS system	mixed data types in expressions
UG-14	PG-281
suspending PV-WAVE on a UNIX	radians to degrees PG-27
system UG-14	-

PV-WAVE Index Xi

scalars to date/time variables	CREATE_HOLIDAYS procedure
R2-238	R1-138, UG-239-UG-240
SEC_TO_DT function UG-233	CREATE_WEEKENDS procedure
STR_TO_DT function UG-229,	R1-140, UG-241
UG-231	cross product, calculating R1-142
string data to date/time variables	CROSSP function R1-142
R2-159	CSV files, generating R1-213
strings to	cube, establishes frame of reference
lower case PG-127	R2-311, R2-332
upper and lower case PG-122	cubic splines
upper case PG-127	interpolation R2-132
three-dimensional to	to smooth contours R2-519,
two-dimensional coordinates	UG-108
UG-119	cursor
VAR_TO_DT function UG-232	controlling position of with TVCRS
CONVOL function R1-125, UG-159,	UG-143
UG-162	for X Windows system UG-A-72
convolution R1-125, UG-159	hot spot of UG-A-73
coordinate systems	manipulating with images R2-217
constructing 3D UG-121	position, reading R1-143
converting from one to another	positioning text with UG-91
R1-21, R1-123, R1-129, R1-538,	selecting default UG-A-72
R1-559, UG-60, UG-193	specifying pattern of UG-A-72
data R2-505	system variable, !C R2-537
device R2-505	CURSOR procedure R1-143, UG-91
graphics UG-59, UG-61	curve fitting
homogeneous UG-115	Gaussian R1-353
in PV-WAVE UG-59	least squares fitting R2-181
normalized R2-512	multiple linear regression R2-26
polar R2-515, UG-89	non-linear least squares R1-147
reading the cursor position R1-143,	polynomial R1-552, R1-555
UG-90	singular value decomposition
right-handed UG-116	method R2-181
screen display UG-138	to a surface R2-172
copying	CURVEFIT function R1-147
color tables R2-67, UG-328	custom color tables, creating R2-293
pixels UG-A-72, UG-A-85	cut-away volumes, defining R2-253
CORRELATE function R1-133	cwavec routine PG-333-PG-340
correlation and regression analysis,	cwavefor routine PG-333-PG-340
Hilbert transform R1-381	calling PV-WAVE PG-334
correlation coefficient, computing for	CYLINDER function R1-150, UG-197
arrays R1-133	
COS function R1-135	
COSH function R1-136	
COSINES function R1-137	

xii PV-WAVE Index

count-intensity distributions R1-386

D	floating-point, with format reversion PG-A-6
!D system variables, fields of R2-537—R2-540 damage repair of windows UG-A-7 data See also data files 3D graphics UG-115 accessing with wavevars PG-350 adding to existing plots UG-66 annotating R2-490 building tables from R1-63 changing to another coordinate system UG-59, UG-119 color of R2-504 column-oriented R1-168, R1-184, R1-207, R1-216 converting byte type to characters PG-125 coordinates UG-193 date/time variables to double-precision variables R1-286 date/time variables to numerical data R1-292 date/time variables to strings R1-289 into date/time UG-229 scalars to date/time variables R2-238 coordinate systems R2-505, UG-59, UG-61 dense R1-320 drop-outs PG-58 export with C format strings PG-A-22, PG-A-24 with FORTRAN format strings PG-A-22 extracting See extracting data fitting, cubic spline R2-132 fixed format I/O PG-152, PG-165, PG-175	formatting into strings PG-124, PG-A-1— PG-A-2 with STRING function PG-199 free format PG-139 ASCII I/O PG-159, PG-161, PG-164 output PG-164 gridding irregular R1-362 importing from more than one file UG-187 with C format strings PG-A-22 with FORTRAN format strings PG-A-22 input/output from a file PG-12 of unformatted string variables PG-197 irregular R1-362 linear regression fit to R2-26 logarithmic scaling UG-84 magnetic tape storage R2-38 manipulation UG-8, UG-191 maximum value to contour R2-510 range R2-526, R2-530, R2-534, R2-556 reading 8-bit image PG-189 for date/time UG-228 from magnetic tapes R2-193 into arrays PG-180, PG-181 tables of formatted data PG-175 unformatted R2-17, PG-193 record-oriented PG-150 reduction before plotting R2-512 row-oriented PG-146 scaling to byte values R1-73 skipping over R2-115 smoothing of R2-119 sorting tables of formatted data PG-175
	sparse R1-368

PV-WAVE Index XIII

three-dimensional scaling R2-57	fixed format ASCII data R1-203
TIFF format PG-191	TIFF R1-221, R1-223
transfer	to a file in ASCII free format
using FORTRAN or C formats	R1-212
PG-165	data types
with XDR files PG-206	byte PG-3
transformation	combining in arrays PG-39-PG-40
by T3D procedure R2-185,	complex PG-3
UG-116	constants PG-17
rotating R2-48, R2-51, UG-117	double precision PG-3
scaling UG-117	floating point PG-3
type conversion PG-125	integer PG-3
types of PG-17	longword PG-3
XDR routines for PG-210	mixed, in
unformatted PG-193	expressions PG-35
user-defined coordinate system	relational operators PG-42
UG-121	seven basics PG-34
writing	string PG-3
date/time UG-255	structure PG-3
to a file PG-12	variables PG-24
to magnetic tape R2-194	date/time
unformatted PG-193	creating empty variables UG-227
unformatted output R2-361	data
with output formats PG-164	adding R1-270
data files	compressing R1-270, R1-272
CSV format R1-213, PG-154	converting to UG-229
logical records PG-147	converting to double-precision
reading	variables R1-286
24-bit image data R1-195	converting to numerical data
8-bit image data R1-192	R1-292
ASCII data from a specific file	converting to string data
R2-17	R1-289
ASCII data from the standard in-	converting to strings for tables
put stream R2-17	UG-282
binary data from a specific file	decrementing values R1-283
R2-17	determining elapsed time
fixed-format ASCII data	R1-277
R1-161	duration R1-277
freely-formatted ASCII data	formats UG-230
R1-177	generating day of the year for
TIFF R1-198	each date R1-155
row-oriented ASCII PG-146	holidays UG-239
writing	in tables UG-281
24-bit image data R1-221	incrementing values R1-270
8-bit image data R1-219	plotting UG-243-UG-253,
ASCII free format data R1-203	UG-267

XIV PV-WAVE Index

printing values R1-282	DC_READ_24_Bit function R1-195,
reading into PV-WAVE PG-170	PG-154
removing holidays and week-	DC_READ_8_BIT function R1-192,
ends R1-272	PG-154
subtracting R1-283	DC_READ_FIXED function R1-161,
templates for transferring	PG-157, PG-174–PG-184, PG-187,
PG-169	PG-A-8, UG-252
transfer with	DC READ FREE function R1-177,
DC READ FIXED PG-174	PG-136, PG-156, UG-246–UG-248
transferring with templates	DC_READ_TIFF function R1-198,
PG-168	PG-154
	DC WRITE 24 BIT function R1-221,
using Julian day for tables	PG-154
UG-283	
using to generate specific arrays	DC_WRITE_8_BIT function R1-219,
R1-279	PG-154, PG-191
weekends UG-241	DC_WRITE_FIXED function R1-203,
writing to a file UG-255	PG-157
DC_WRITE functions UG-255	DC_WRITE_FREE function R1-212,
description of UG-221	PG-137, PG-156
excluding days UG-239	DC_WRITE_TIFF function R1-223,
Julian day UG-225	PG-154, PG-191
reading data UG-228	DCL
recalc flag UG-226	logical names R2-74
structures	symbols
elements of UG-225	defining R2-74
importing data into PG-162	deleting R1-238
templates PG-169	deallocating
variables	file units R1-344
creating from scalars R2-238	example R1-348, PG-222
with current system date and	LUNs PG-141, PG-143
time R2-199	DEC
DAY NAME function R1-153, UG-259	printers UG-A-58
DAY OF WEEK function R1-154,	terminals, generating output for
UG-260	UG-A-54
DAY OF YEAR function R1-155,	decal, definition of UG-200-UG-201
UG-261	declaring variables PG-5
DBLARR function R1-156	decomposed color UG-A-79
DC (Data Connection) functions	DECStation 3100, error handling
	PG-268
advantages of PG-153	DECW\$DISPLAY logical, for X Windows
description of PG-13	UG-A-69
for simplified data connection	
PG-153	DECwindows Motif UG-A-69
opening, closing files PG-140	DEFINE_KEY procedure R1-226
table of PG-14	DEFROI function R1-232
DC_ERROR_MSG function R1-157	DEFSYSV procedure R1-236
DC_OPTIONS function B1-159	

PV-WAVE Index XV

degrees	differentiation, numerical R1-246
convert from radians PG-27	diffuse component of color, for RENDER
convert to radians R2-541, PG-27	function UG-198
delaying program execution R2-274	digital gradient function PG-237
DELETE SYMBOL procedure R1-238,	DIGITAL_FILTER function R1-250
PG-297	DILATE function R1-254
deleting	dimensions of expressions, determining
compiled functions from memory	PG-272
R1-239	DINDGEN function R1-259
compiled procedures from memory	!Dir system variable R2-540
R1-241	directory path
DCL symbols R1-238	!Path UG-20
ERASE procedure R1-298	searching for procedures and func-
variables R1-244	tions R2-550
VMS logical names R1-240	directory stack
VMS symbols R1-238	popping directories off of R1-578
DELFUNC procedure R1-239	printing out R1-580
DELLOG procedure R1-240, PG-296	pushing R1-590
DELPROC procedure R1-241	directory, changing to R1-79
DELSTRUCT procedure R1-242,	disappearing variables PG-251, PG-258
PG-104	display
DELVAR procedure R1-244, PG-104	color, control of UG-145, UG-310
demonstration gallery UG-179	dithering UG-148
demonstration programs UG-183	of images R2-225
dense data R1-320	of multiple plots on a page UG-83
density function, calculating histogram	reading from R2-223, UG-142
R1-386, UG-329	\$DISPLAY environment variable, for X
DERIV function R1-246	Windows UG-A-69
Desc qualifier, for sorting columns	!Display_Size system variable R2-540
UG-276	DIST function R1-260, UG-163
DETERM function R1-248	Distinct qualifier, for removing duplicate
determinant, calculating R1-246,	rows in tables UG-272
R1-248	distortion, linear UG-170
device	dithering, different methods compared
controlling output of R1-249	UG-148
coordinate systems UG-59, UG-61	division operator (/) PG-32
current parameters R2-537	DOC_LIBRARY procedure R1-264
name of R2-539	documentation of user routines R1-264
selecting output from R2-66	dollar sign
device drivers	as continuation character UG-34
general information about UG-A-1	format code PG-A-10, PG-A-12
getting information about PG-398	in command files UG-21
selecting output from UG-A-2	DOUBLE function R1-267
table of supported UG-A-2	double-buffering UG-A-79
DEVICE procedure R1-249, UG-A-4	
dialog box R2-419, PG-453-PG-454	

xvi PV-WAVE Index

double-precision	editor
arrays, creating R1-156, R1-259	See also text widget
constants PG-20	creating text PG-445
data type PG-17	eigenvalues and eigenvectors, determin-
data, converting to R1-267	ing R2-201
variables, converting date/time vari-	8-bit image data
ables to R1-286	how stored PG-189
!Dpi system variable R2-540	writing data to a file R1-219
drawing area R2-422, PG-439	8-bit color UG-A-79
driver	elements
default settings	definition of PG-4
SIXEL UG-A-58	finding the number of R1-469
See device drivers	ELSE
	in CASE statement PG-62
DROP_EXEC_ON_SELECT procedure	in IF statement PG-71
R1-269	empty output buffer R1-294, R1-344
DT_ADD function R1-270, UG-237	
DT_COMPRESS function R1-272,	EMPTY procedure R1-294
UG-242	Encapsulated PostScript
DT_DURATION function R1-277,	Interchange Format UG-A-34
UG-239	keyword UG-A-33
DT_PRINT procedure R1-282, UG-261	with Microsoft Word UG-A-46
DT_SUBTRACT function R1-283,	END identifier PG-60—PG-61
UG-238	end of file
DT_TO_SEC function R1-286, UG-258	testing for R1-296
DT_TO_STR procedure R1-289,	when applied to a pipe PG-310
UG-256	writing to tape R2-276
DT_TO_VAR procedure R1-292,	ENDCASE, end statement PG-62
UG-257	ENDIF, end statement PG-72
DTGEN function R1-279	ending PV-WAVE sessions PG-315,
!Dtor system variable R2-541	PG-335, PG-341
dynamic	ENDREPEAT, end statement PG-61
data types PG-35	ENDWHILE, end statement PG-61
structures PG-35	enhancing contrast, of images R1-383
	environment
E	after an error PG-250
-	manipulating R1-77
eavesdrop mode, HPGL plotter output	modifying UG-44
UG-A-18	ENVIRONMENT function R1-295,
edge enhancement	PG-294
Roberts method R2-45	environment variables
Sobel method R2-122	\$DISPLAY for X Windows UG-A-69
	adding R2-64, PG-293
!Edit_Input system variable R2-541	changing R2-64, PG-292
editing, command line UG-33	DELETE SYMBOL procedure
	(VMS) PG-297
	DELLOG procedure (VMS) PG-296
	52220 a procoduro (************************************

PV-WAVE Index XVII

ENVIRONMENT function (UNIX)	during program execution PG-258
PG-294	enabling/disabling math traps
GETENV function (UNIX) PG-294	PG-266
obtaining	example of checking for PG-267
all R1-295	floating-point PG-263
values of R1-356, PG-294	for I/O PG-140
SET_SYMBOL procedure (VMS)	function, evaluating R1-304
PG-296	input/output R1-479, PG-259
SETENV procedure (UNIX) PG-293	list of procedures PG-257
SETLOG procedure (VMS) PG-295	machine dependent handling
translating PG-294	PG-268
TRNLOG function (VMS) PG-295	math
UNIX PG-292	See math errors
VMS PG-295-PG-296	mechanism in procedures and func-
WAVE_DEVICE UG-44, UG-45	tions PG-250
WAVE DIR UG-45	MESSAGE procedure PG-251
WAVE_STARTUP UG-48, UG-51	messages PG-261—PG-262
EOF	prefix for R2-542
function R1-296, PG-171, PG-219,	setting report level for "DC" func-
PG-310	tions R1-159
statement, example of PG-78	obtaining text of message R2-541
testing for PG-219	ON_ERROR procedure PG-258
WEOF procedure, writing to end of	ON_IOERROR procedure PG-259
mark on tape PG-229	options for recovery PG-258
writing to tape R2-276	overflow PG-264
EQ operator	recovery from PG-250, PG-259
description of PG-48	running SunOS Version 4 PG-268
operator precedence PG-33	Sun-3 and Sun-4 PG-268
equality	VMS PG-268
See EQ operator	ERRORF function R1-304
ERASE procedure R1-298	ERRPLOT procedure R1-305
erasing	escape sequences
screen of graphics device R1-298	creating PG-22
variables UG-27	SIXEL output UG-A-61
ERODE function R1-300	event handler
!Err system variable R2-541	definition R2-367, PG-487
!Err_String system variable R2-541	registering R2-367, PG-487
error bars, plotting R1-305	event loop
error handling	WAVE Widgets PG-471
"Plot truncated" message R2-515	Widget Toolbox PG-494
See also math errors	exclamation point UG-34
arithmetic PG-263	excluding days from date/time variables
categories of PG-257, PG-258	UG-239
CHECK_MATH function R1-90,	exclusive OR operator PG-50
PG-263	EXEC_ON_SELECT procedure R1-309
DECStation 3100 PG-268	2/120_0/1_0222201 procedure 1/1-009

xviii PV-WAVE Index

EXECUTE function R1-313, PG-243—	explicitly formatted
PG-244, PG-273	See fixed format
executing	exponential function, natural R1-317
command files UG-19	exponentiation operator (^)
existing functions and procedures	examples PG-46
UG-23	operator precedence PG-32
interactive programs UG-25	exporting data PG-191
main programs UG-24	exposing windows R2-363
operating system commands	expressions
R2-126	data type and structure evaluation
statements one at a time R1-313	PG-34
executive commands	description of PG-6
LOCALS compiler directive	efficiency of evaluation PG-279
PG-244	evaluation of PG-35
.CON UG-15, UG-29	finding attributes of PG-272
.GO UG-29	forcing to specific data types PG-36
LOCALS PG-243, UG-31	general description PG-31
.RNEW UG-28, UG-29	invariant PG-281
.RUN	mixed data types in expressions
compiling functions and proce-	PG-35
dures PG-239	overview of PG-6
description UG-27	structure of PG-34, PG-39
examples UG-29	type of PG-34
-I argument UG-28	with arrays PG-40
syntax PG-239	with mixed data types PG-281
-t argument UG-28	Extended Metafile System UG-A-13
.SIZE PG-242-PG-243, UG-31	extracting data
.SKIP UG-30	See also converting data to
.STEP UG-30	double-precision floating-point data
definition of UG-26	type R1-267
table of UG-27	from variables PG-37, PG-38
EXIT procedure R1-316, PG-335,	Julian day numbers R1-421
PG-341, UG-14	longword integer type R1-442
exiting	string type R2-144
an application	extracting image plane UG-A-95
See closing widget hierarchy	
PV-WAVE	F
EXIT procedure, description	•
R1-316	false
on a UNIX system UG-14	definitions for different data types
on a VMS system UG-14	PG-71
EXP function R1-317	representation of PG-42
expanding	Fast Fourier Transform R1-328
color tables R2-142	FAST GRID2 function R1-319, UG-191
images R2-21, R2-493	FAST GRID3 function R1-322, UG-191
·	·-·

PV-WAVE Index XIX

FAST_GRID4 function R1-325, UG-191 FFT function	procedures for fixed format I/O PG-165
applied to images UG-162	reading R2-17
description R1-328	
fields	C programs PG-196
definition of PG-101	DC_READ_FIXED_PG-177—
extraction and conversion of PG-37	PG-184, PG-187, PG-A-8
reference to PG-108	FORTRAN binary data with PV-WAVE PG-224
syntax PG-108	
file selection box (widget) R2-426,	one input character R1-358 READF PG-175-PG-185
PG-456—PG-458	READU PG-175—FG-185
FILEPATH function R1-331	unformatted R2-17
files	record-oriented binary files PG-149
access mode, VMS PG-225	row-oriented PG-146
allocating units R1-359, PG-139,	skipping over R2-115
PG-143	standard input, output and error
closing R1-94, PG-140	PG-140
column-oriented ASCII PG-144	startup command UG-48
compression of TIFF files PG-192	stopping batch R2-138
CSV format PG-154	testing for end of file R1-296,
deallocating LUNs R1-345,	PG-219
PG-141, PG-143	units, closing R1-94
formatting, codes for PG-A-9-	VMS
PG-A-25	attributes PG-227
free format ASCII I/O PG-156	fixed-length record format
FREE LUN procedure, deallocating	PG-217, PG-226
files PG-143	record oriented PG-226
getting information about PG-220,	when to open PG-140
PG-399	writing
header, example PG-A-3	8-bit image data to R1-219,
I/O with structures PG-116	R1-221
in UNIX PG-223	ASCII free format to R1-203,
indexed files for VMS PG-228	R1-212
inputting data from PG-12	data to PG-12
locating on disk PG-218	unformatted output R2-361
logical records, changing size of	with UNIX FORTRAN programs
PG-147	PG-200
logical units PG-138	with WRITEU PG-195
on magnetic tape R2-38	XDR PG-205
opening R1-480, PG-138-PG-139	filling
organization of VMS PG-224	contour plots UG-110
organization options PG-144	pattern R2-506
pointer, positioning R1-519,	filters
PG-219	2D UG-163
portable binary PG-205	bandpass UG-164
printing R1-580	Butterworth UG-164

XX PV-WAVE Index

creating digital filters R1-250	fonts
exponential high or low pass	3D transformations UG-292
UG-164	appearance (hardware vs. soft-
for	ware) UG-291
images UG-162-UG-163	changing UG-295
mean smoothing R2-119	character size R2-501, R2-518
median smoothing R1-454	computer time for drawing UG-293
signals or images R1-260	flexibility (hardware vs. software)
high pass UG-160, UG-164	UG-293
low pass UG-164	formatting commands UG-293
Roberts UG-160	hardware R2-507
Sobel UG-160	hardware vs. software UG-291
FINDFILE function R1-333, PG-218	Hershey character sets UG-291
FINDGEN function R1-335	index of
FINITE function R1-336, PG-36,	graphics text font R2-544
PG-264	hardware font R2-544
finite values, checking for R1-336	list of PG-464
fitting with Gaussian curve R1-353	portability (hardware vs. software)
FIX function	UG-292
description R1-338	positioning commands UG-301
example PG-36-PG-37	PostScript
fixed format PG-156	See PostScript fonts
ASCII I/O PG-155, PG-157	selecting R2-507
comparison with free format	to annotate plots UG-68
PG-152	selection commands UG-295
data, examples of reading PG-176	setting in WAVE Widgets PG-464
I/O PG-165, PG-175	software R2-507, R2-563
fixed-length record format, VMS	specifying R2-507
PG-217, PG-226	vector-drawn R2-507, UG-291
FLOAT function R1-340, PG-37	with graphic routines R2-507
example PG-21	with plotting routines R2-507
floating-point	FOR statement
arrays, creating R1-340, R1-343	definition PG-9
constants PG-20	examples PG-65, PG-66
data type PG-3, PG-17	(variable increment) PG-67
errors PG-263	explicit increment PG-67
format code defaults PG-A-14	format with a block statement
type, converting to R1-340, R1-343	PG-60
Floyd-Steinberg dithering UG-148	two forms of PG-65
FLTARR function R1-343	force fields
flush output buffer R1-294, R1-344	examples PG-65
FLUSH procedure R1-344, PG-158,	increment parameter PG-65
PG-219	plotting R1-510, R2-244, R2-248
flushing file units PG-218	simple PG-65
Font field of !P UG-293	•

PV-WAVE Index XXI

form layout (widgets)	format strings PG-A-8
See also layout (WAVE widgets)	reading multiple array ele-
PG-422	ments PG-185
formal parameters	programs
copying actual parameters into	calling PV-WAVE (wavecmd)
PG-236	PG-318
definition of PG-235	linking to PV-WAVE under UNIX
format	PG-348
C and FORTRAN for data transfer	reading
PG-165	data in relation to UNIX
for	PG-199
DC (Data Connection) functions PG-169	multiple array elements PG-185
writing data PG-164	sending commands to PV-WAVE
interpreting format strings PG-166	wavecmd PG-315
tick labels UG-81-UG-82	writing data to PV-WAVE PG-181-
format codes	PG-185
C PG-A-22	4D gridding R1-325, R1-369, UG-191
examples of integer output PG-A-17	Fourier 71-323, A1-369, 00-191
floating-point	
defaults PG-A-14	spectrum, 2D UG-165 transform
output PG-A-15	
FORTRAN PG-A-9-PG-A-21	See FFT function
group repeat specifications PG-A-7	free format
	ASCII I/O PG-155—PG-156
integer defaults PG-A-16	input PG-159—PG-160
format reversion PG-167, PG-A-5	output PG-164
format strings	FREE_LUN procedure R1-345,
C, for data import and export	PG-143–PG-144
PG-A-22	frequency
description of PG-A-1	domain techniques UG-162
FORTRAN PG-185, PG-A-22	image UG-163
when to use PG-A-2	FSTAT function R1-346, PG-220-
formatted data	PG-221
commands for software fonts	FUNCT procedure R1-350
UG-293	FUNCTION definition statement PG-235
rules for PG-159, PG-165	function keys
strings PG-124	defining R1-226
structures for I/O PG-116	equating to character strings UG-52
using STRING function PG-187	for line editing UG-33
FORTRAN	getting information about R1-410,
binary data, reading with	PG-400
PV-WAVE PG-224	functions
ending PV-WAVE with waveterm	actual parameters PG-234
PG-315	calling PG-11
format codes PG-A-9-PG-A-21	calling mechanism PG-248

xxii PV-WAVE Index

checking for	search path UG-46-UG-50
keyword parameters R1-422	syntax for
number of elements R1-469	calling PG-15
parameters PG-269	defining PG-15
positional parameters R1-471	type conversion PG-36
compiling PG-239	user-defined PG-241
automatically PG-69, PG-239	
with .RUN and filename	\boldsymbol{G}
PG-239	G
with interactive mode PG-240	GAMMA function R1-352
conditions for automatic compila-	GAUSSFIT function R1-353
tion PG-234	Gaussian
copying actual parameters into for-	curve fitting R1-353
mal parameters PG-236	
creating PG-15	function, evaluating R1-355
	integral R1-355
interactively UG-26	GAUSSINT function R1-355
with a text editor UG-22	GE operator
definition of PG-233	description of PG-49
definition statement of FUNCTION	for masking arrays PG-49
PG-235	operator precedence PG-33
deleting from memory R1-239	geometric transformations UG-167,
determining number of parameters	UG-168
PG-269	GET_KBRD function R1-358, PG-154,
directory path for R2-550, PG-28	PG-222, PG-223
documentation of user R1-264	GET_LUN procedure R1-359, PG-143
error handling PG-250	GET_SYMBOL function R1-361
formal parameters PG-235	GETENV function R1-356, PG-294
I/O errors in R1-479	GOTO statement PG-10, PG-52, PG-70
I/O for simplified data connection	Gouraud shading R1-564, R2-70-
PG-13	R2-71, UG-131
information, obtaining with INFO pro-	graphical user interface
cedure PG-401	See GUI
keyword parameters PG-68, PG-74	graphics
libraries of VMS PG-251	2D array as 3D plot R2-197
library UG-23	3D to 2D transformations R2-549
maximum size of code PG-397	bar charts UG-76
number of parameters required	changing graphics device UG-45
PG-236	clipping R2-503
overview of PG-2, PG-14	output R2-503
parameters PG-74, PG-234	window R2-543
passing mechanism PG-246	combining
positional PG-68, PG-74,	CONTOUR and PLOT proce-
PG-235	dures UG-127
program control routines PG-273	CONTOUR and SURFACE pro-
recursion PG-248	cedures UG-126
required components of PG-237	

PV-WAVE Index XXIII

connecting symbols with lines	text
R2-548	See annotation
converting from 3D to 2D coordinates	thickness of lines R2-522
UG-119	3D UG-115
coordinate systems for UG-59,	three-dimensional scaling R2-57
UG-116	tick marks
current device parameters R2-537,	See tick
PG-398	title R2-550
cursor R1-143, UG-90	transformation matrices R2-185,
device driver	UG-116
changing UG-45	window display UG-A-6
controlling R1-249	graphics window (drawing area) PG-439
size of display R2-539	gray
supported by PV-WAVE	levels
UG-A-2	dithering UG-148
establishing a 3D coordinate system	transformations UG-152
UG-121	scales PG-189, PG-192
exposing and hiding windows	greater than
R2-363	See GT operator
extracting image profiles R1-583	greater than or equal
keywords R2-497	See GE operator
line thickness R2-522, R2-549	GRID function R1-362
multiple graphs on a page R2-545	GRID_2D function R1-364, UG-191
output device	GRID_3D function R1-367, UG-191
configuring with DEVICE	GRID_4D function R1-369, UG-191
UG-A-4	GRID_SPHERE function R1-373,
selecting with SET_PLOT	UG-192
UG-A-3	gridding
overlaying an image on a contour	2D R1-319, R1-364
R1-402, UG-100	3D R1-322, R1-367
polygon	4D R1-325, R1-369
filling R1-542, UG-74	definition of UG-191
shading R1-564	spheres R1-373
positioning in window R2-547	summary of routines R1-22
procedures for plotting data UG-55	table of UG-184
reading	with dense data points R1-319,
cursor position R1-143	R1-322, R1-325
values from cursor R2-15	with sparse data points R1-364,
shaded surfaces R2-83, R2-91,	R1-367, R1-369
UG-131	grids
specifying range of data to plot	making plots with R1-362, UG-81
R2-556	plotting keyword for UG-78
speeding up plotting R2-547	Group By clause UG-273-UG-275
subtitle R2-549	group repeat specifications PG-A-6-
symbols R2-547	PG-A-7
	groups of statements PG-60

xxiv PV-WAVE Index

GT operator description of PG-49	HIST_EQUAL_CT procedure R1-386, UG-156
operator precedence PG-33	histogram
GUI	calculating density function R1-386,
methods of creating PG-408	UG-329
selecting look-and-feel PG-411,	equalization R1-383, UG-155
UG-51	HISTOGRAM function R1-388,
	UG-155
ப	mode UG-70
T 1	of volumetric surface data R2-312
HAK procedure R1-377	HLS
handler, event R2-367, PG-487	color model UG-308, UG-309
HANNING function R1-378	compared to HSV UG-308
hardcopy devices	procedure R1-396, UG-329
See output devices	!Holiday_List system variable R1-437
hardware fonts	holidays, removing from date/time vari-
See fonts	ables with DT COMPRESS
hardware pixels UG-A-96	R1-272
hardware polygon fill UG-A-18	homogeneous coordinate systems
help	UG-115
See also information	HPGL output UG-A-13-UG-A-19
documentation of user-written rou-	HSV
tines R1-264	color model UG-308, UG-309
getting R1-410	compared to HLS UG-308
HELP procedure R1-410	procedure R1-398, UG-329
Hershey fonts R2-507, UG-291	HSV_TO_RGB procedure R1-400
Hewlett-Packard	hyperbolic
Graphics Language plotters	cosine R1-136
UG-A-13	sine R2-112
ink jet printers UG-A-23	tangent R2-191
laser jet printers UG-A-23	
Printer Control Language printers	1
See PCL output	•
hexadecimal characters, for representing	icons
non-printable characters PG-23	on menu PG-431
hexadecimal value, specifies color	on tool box PG-433
UG-A-91	turning windows into R2-363
hide a widget PG-469	IF statement PG-70—PG-72
hiding windows R2-363	avoiding PG-276, PG-277
hierarchy of operators PG-32	definition PG-9
high pass filters R1-250, UG-160,	image processing
UG-164	See also images
HILBERT function R1-381	calculating histograms R1-386
HIST function, example of PG-66	convolution R1-125
HIST_EQUAL function R1-383, UG-158	creating digital filters R1-250

PV-WAVE Index XXV

edge enhancement R2-45, R2-122	dilation operator R1-254
Fast Fourier Transform R1-328	direction of display (!Order) R2-542
Hanning filter R1-378	display
histogram equalization R1-383	order UG-138
magnifying images R1-125, R2-21	routines UG-4, UG-136
minimizing images R2-21	TVRD function R2-223
morphological dilation R1-254	without scaling intensity
morphological erosion R1-300	,
	R2-212
polynomial warping R1-526,	dithering UG-148
R1-574	8-bit format PG-189
Roberts edge enhancement R2-45	erosion operator R1-300
rotating images R2-48, R2-51	expanding R1-115, R2-24, R2-36
selecting a region of interest	extracting plane UG-A-95
R1-232	extracting profiles from R1-583
smoothing R1-454, R2-119	Fast Fourier Transform R1-328
Sobel edge enhancement R2-122	filtering UG-162
write mask UG-328	frequency domain techniques
IMAGE_CONT procedure R1-402	UG-162
images	geometric transformations UG-167
See also color, image processing	interleaving R1-197, R1-201,
24-bit, reading R1-195	R1-222
8-bit, reading R1-192	interpolation of UG-170
animating R2-277, R2-315	•
annotation of R2-490	magnifying R2-24, R2-36, R2-54,
	R2-225, R2-493, UG-140
as pixels PG-192	modifying intensities of UG-152
combining	morphologic dilation of R1-254
CONTOUR and SURFACE pro-	morphologic erosion of R1-300
cedures UG-126	optimizing transfer of UG-A-82
with surface and contour plots	orientation of UG-138
R2-104	overlaying with contour plots
with three-dimensional graph-	R1-402, UG-100
ics UG-129-UG-130	placing the cursor in UG-143
contrast	polynomial warping of R1-521,
control UG-331	UG-168
enhancement UG-330	position of on screen UG-138
convolution of R1-125, UG-159	PostScript display of UG-A-39
data	profiles R1-583
how stored PG-189	reading
interleaving PG-190, PG-193	from display device UG-142
output PG-191	pixel values from R1-550
pixels PG-193	reformatting of R1-574
reading block of from tape	resizing of R2-225
PG-231	
	Roberts edge enhancement R2-45
reading with associated variable	rotating R2-54, UG-168
method PG-213	routines used to display UG-135
definition of UG-135	scaling to bytes UG-154

XXVI PV-WAVE Index

sharpening UG-160	information, obtaining with INFO proce-
shrinking R1-115	dure R1-410
or expanding R2-21, UG-140	informational procedures, list of PG-257
size of display UG-139	initializing WAVE Widgets PG-413
smoothing R2-119	input/output R1-479
Sobel edge enhancement of	advantages and disadvantages of
B2-122	each type PG-152
spatial warping R1-574	ASCII PG-151
special effects R1-543, R1-561,	ASSOC with unformatted data
UG-328, UG-A-78	PG-212
subscripts of pixels inside polygon	binary data PG-151, PG-188
region R1-550	C binary data PG-194
transformation matrices UG-116	C-generated XDR data PG-207
transposing of R2-204	date/time data PG-162, PG-170
true-color UG-146—UG-147	DC (Data Connection) functions
24-bit format PG-189	PG-140, PG-153
value of individual pixels R2-327	efficiency of associated variables
warping UG-168	PG-212
with contour plots UG-100	error handling R1-479, PG-140,
zooming of R2-493	PG-259
IMAGINARY function R1-405	fixed formats PG-152, PG-165
example PG-21	flushing file units PG-218
IMG TRUE8 procedure R1-406	format reversion PG-167
In operator UG-280	free format R1-580, R2-17,
include files for widgets PG-495	PG-159, PG-164
-	from
	keyboard R1-358, PG-222
inclusive OR operator PG-50	word-processing application
increment parameter	PG-175
error if equal to zero PG-67	image data PG-189
in FOR statement PG-67	into
index number, color index UG-306,	complex variables PG-160
UG-322	structures PG-161
indexed files, VMS PG-228	overview of PG-12
INDGEN function	portable binary PG-206
description R1-409	procedures
using to make color tables UG-313	for ASCII data PG-155
indices of colors in color table R2-294	for binary data PG-154
indirect compilation UG-21	READF procedure PG-175
Infinity PG-263	reading records with multiple array
INFO procedure	elements PG-180-PG-182,
description of R1-410, PG-395,	· · · · · · · · · · · · · · · · · · ·
UG-16	PG-185
getting information about files	rules for
PG-220	binary transfer of string vari-
viewing table structure UG-268	ables PG-197
	fixed format PG-156

PV-WAVE Index XXVII

formatted data PG-159,	interleaving
PG-165	description of PG-190
record-oriented binary files	image data PG-190, PG-193
PG-149	in 24-bit images R1-197, R1-201,
selecting type of PG-138	R1-222
strings with structures PG-117	pixels PG-192
structures PG-116	INTERPOL function R1-414
types of PG-151	interpolated shading UG-131
unformatted R2-17	interpolation
using UNIX pipes PG-310	bilinear R1-57
VMS binary files PG-149	cubic spline R2-132
waiting for input R2-61	linear, between colortable values
when to open PG-140	R2-294
XDR files PG-205	interprocess communication
INTARR function R1-413	See interapplication communication
integer	invariant expressions, removing from
bit shifting R1-418	loops PG-281
constants	INVERT function R1-416
examples PG-19	inverting pixels UG-A-95
range of PG-19	irregular data, gridding R1-362
syntax of PG-18	ISHFT function R1-418
conversions, errors in PG-264	iso-surfaces
data	example of UG-212, UG-214,
shown in figure PG-146	UG-216
type PG-3, PG-17	viewing interactively R2-307
writing with format reversion	,
PG-A-5—PG-A-6	J
format defaults PG-A-16	U
output, for format codes PG-A-17	JOURNAL procedure R1-419
syntax of constants PG-18	!Journal system variable R2-542
type array, creating R1-413	journaling R1-419
type, converting to R1-338	description of UG-36
Integral of Gaussian R1-355	examples UG-38
intensities, modifying images UG-152	in relation to PRINTF UG-37
ntensity, in 24-bit color UG-A-91	obtaining unit number of output
nterapplication communication	R2-542, PG-28
bidirectional and unidirectional	JUL_TO_DT function R1-421
PG-307	description of UG-233
calling PV-WAVE from a C pro-	example of UG-234, UG-253
gram PG-337	Julian day
methods of PG-305	converting to a date/time variable
using SPAWN PG-309	R1-421
with a child process PG-309	description of UG-225
with RPCs (remote procedure	in IDT Base R2-60
calls) PG-359	using with tables UG-283
	309

xxviii PV-WAVE Index

K	LaTeX documents
	inserting plots UG-A-40
keyboard	using PostScript with UG-A-40,
accelerators UG-52	UG-A-43
defining keys R1-226, UG-52	layout (WAVE Widgets)
getting input from R1-358, PG-222	arranging a PG-422
interrupt UG-15, UG-29	example PG-417, PG-420-PG-421
key definitions PG-400	form PG-422
	row/column PG-420
line editing, enabling R2-541, PG-28	LE operator
	description of PG-49
using for command recall UG-32	operator precedence PG-33
KEYWORD_SET function R1-422,	least square
PG-238, PG-239, PG-269	curve fitting R1-552, R1-574,
keywords	UG-72
checking for presence of PG-271	non-linear curve fitting R1-147
description of UG-23	problems, solving R2-179
examples PG-74	Lee filter algorithm R1-424
with functions PG-238	LEEFILT function R1-424
parameters	LEGEND procedure R1-426
abbreviating PG-74 advantages of PG-75	legend, adding to a plot R1-426
and functions PG-68	less than
checking for presence of	See LT operator
R1-422, PG-239, PG-269	less than or equal
definition of PG-74, PG-235	See LE operator
graphics and plotting routines	libraries
R2-497	creating and revising for VMS
passing of PG-234, PG-235	PG-253
using the Keyword construct	PV-WAVE Users' PG-255
PG-74, PG-235	searching (VMS) PG-252
relationship to system variables	Standard PG-255
UG-24, UG-57	light source
Korn shell PG-299	lighting model, for RENDER function UG-197
•	modifying R2-70
_	shading R1-564, R2-70, UG-131
to to the destinations of COTO statements	setting parameters for R2-70
labels, destinations of GOTO statements	LINDGEN function R1-427
PG-52	line
Lambertian	color of R2-504
ambient component UG-199	connecting symbols with UG-72
diffuse component UG-198	drawing R1-510, UG-121
transmission component UG-199	fitting, example using POLY_FIT
landscape orientation UG-A-34	UG-72
	linestyle index R2-545

PV-WAVE Index XXIX

style of R2-502, R2-545	LOAD_HOLIDAYS procedure R1-437
thickness of R2-522, R2-549	UG-240
linear algebra	LOAD_WEEKENDS procedure R1-439 UG-241, UG-242
eigenvalues and eigenvectors	LOADCT procedure R1-435, UG-145,
R2-201	UG-311, UG-329
LU decomposition R1-445,	description UG-136
R1-447	local variables, definition of PG-237
reducing matrices R2-207	log scaling, plotting R1-497
rules PG-46	logarithm
solving matrices R2-207,	base 10 R1-40
R2-208	natural R1-38
axes, specifying R2-560	logarithmic
distortion UG-170	axes R1-497
equations	specifying R2-560
solution vector R1-468	plotting R1-497
solving R2-177	scaling R1-497
least squares problems, solving	keywords UG-84
R2-179	PLOT_IO procedure UG-85
regression, fit to data R2-26	logical names
linking	defining R2-65
applications to PV-WAVE PG-346	VMS, DCL R2-209
C code to PV-WAVE R1-429	VMS, deleting R1-240
C program under	logical operators
UNIX PG-347	evaluating PG-42
VMS PG-349	representation of PG-42
client with PV-WAVE PG-361	used with arrays PG-42
FORTRAN applications under	logical records PG-147
UNIX PG-348	logical unit number
LINKNLOAD function	allocating R1-359
accessing external functions	See LUNs
R1-433, PG-323, PG-327	LONARR function R1-440
calling external programs	LONG function R1-442
PG-319	longword
compiling under UNIX PG-322,	data type PG-17
PG-326, PG-328	integer arrays, creating R1-427,
description R1-429, PG-319	R1-440
example R1-431, PG-321,	integer, converting to R1-442
PG-324	look-and-feel, of GUI PG-411, UG-51
operating systems supported	lookup table, color UG-308, UG-A-79
under PG-320	See also color tables
list, scrolling	loop, event
See scrolling list	See event loop
LJ-250 output UG-A-20—UG-A-23	low pass filters UG-159, UG-164
LN03 procedure R1-434	creating R1-250

XXX PV-WAVE Index

LT operator	skipping
description of PG-49	backward on a tape PG-231
operator precedence PG-33	forward on the tape (VMS)
LUBKSB procedure R1-445	PG-230
LUDCMP procedure R1-447	TAPRD procedure PG-229
LUNs	TAPWRT procedure PG-229
closing PG-140	WEOF procedure PG-229
deallocating R1-345, PG-141	writing to R2-194, PG-229
description PG-138	magnifying images R2-21, R2-36,
for	R2-54, R2-493, UG-140
opening files PG-139	main programs
standard I/O PG-140	definition of UG-24
getting information using FSTAT	difference from command files
PG-220	UG-25
of current output file R2-539	executing UG-24
of journal output R2-542, PG-28	main PV-WAVE directory R2-540,
operating system dependencies	PG-27
PG-142	main window
range of PG-141	See shell window
general use PG-143	MAKE ARRAY function R1-449
reserved numbers PG-140, PG-141	makefile, using PG-346
standard	management, of widgets PG-484
error output PG-142	mapping, of widget PG-484
input PG-141	margin, around plot R2-555
output PG-141	marker symbols
used by	displaying in a volume R2-263
FREE LUN PG-143	for data points UG-72
GET LUN PG-143	user-defined UG-73
with	masking
UNIX PG-142	arrays PG-49
VMS PG-142	unsharp UG-161
VIII.0 1 0 1 12	math errors
A <i>A</i>	See also error handling
IVI	accumulated R1-90
Masintash samputara	accumulated math error status
Macintosh computers	PG-263
See PICT output	CHECK MATH function PG-265
magnetic tape	detection of PG-263
accessing under VMS PG-229	hardware-dependent PG-268
mounting a tape drive (VMS)	procedures for controlling PG-258
PG-230	math messages, issuing R1-460
reading from R2-193, PG-231	math traps, enabling/disabling PG-266
REWIND procedure PG-229	mathematical function
SKIPF procedure PG-229	abbreviated list of UG-9
	absolute value R1-30
	arc-cosine R1-31
	arc-cosine III-oi

PV-WAVE Index XXXI

arcsine R1-41	standard deviation R2-136
arctangent R1-45	tangent R2-190
area of polygon R1-522	mathematical morphology UG-173
base 10 logarithm R1-40	matrices
Bessel I function R1-51	reading and printing PG-93
Bessel J function R1-53	reading interactively R2-40
Bessel Y function R1-55	transformation, See transformation
bilinear interpolation R1-57	matrices
bit shifting R1-418	using subscripts with PG-82
checking for finite values R1-336	matrix
complex conjugate R1-118	expressions PG-99
convolution R1-125	multiplication PG-94
correlation coefficient R1-133	print to specified file unit R1-517
cosine R1-135	print to standard output stream
cross product R1-142	R1-515
cubic splines, interpolation R2-132	reading from a file R2-43, PG-96
derivative R1-246	subarrays PG-98
determinant of matrix R1-248	matrix inversion
error function R1-304	REVERSE function R2-36
Fast Fourier Transform R1-328	SVD procedure R2-179
GAMMA function R1-352	matrix multiplication operator (#)
Gaussian integral R1-355	examples PG-46
Hilbert transform R1-381	operator precedence PG-32
hyperbolic	MAX function R1-452
cosine R1-136	!Max_Levels system variable R2-551
sine R2-112	maximum operator (>)
tangent R2-191	examples PG-46
imaginary numbers R1-405	operator precedence PG-32
improve solution vector R1-468	mean of an array R2-136
LU decomposition R1-445, R1-447	mean smoothing UG-159
matrix	median
reduction R2-207	filter R1-454
solutions R2-208	value of array R1-454
maximum value R1-452	MEDIAN function R1-454, UG-159
mean R2-136	median smoothing UG-159
minimum value R1-462	memory
natural exponent R1-317	allocation PG-244
natural logarithm R1-38	deleting
polynomial functions R1-521	compiled functions from
polynomial roots R2-495	R1-239
random numbers R2-13	compiled procedures from
sine R2-109	R1-241
singular value decomposition	structure definitions R1-242
R2-179	getting information about R1-410
solving R2-177	55 mg 410
square root R2-134	

XXXII PV-WAVE Index

order and arrays PG-281, PG-282	modulo operator PG-49
virtual	monochrome
See virtual memory	devices UG-148
menus	displays UG-A-94
bar PG-426	dithering UG-148
callback PG-428	MONTH_NAME function R1-465,
creating R2-221, R2-359	UG-260
creating and handling PG-425	morphologic
defining text PG-429	dilation operator R1-254
example PG-431	erosion operator R1-300
option PG-428	morphology, mathematical UG-173
popup PG-427	Motif GUI PG-479
	mouse
separators PG-430	accesses text editing functions
using unnamed structure to define	R2-351
text PG-429	positions slice through data R2-332
MESH function	rotates color table R2-286
description of R1-457	
example of UG-204, UG-206	use to rotate and flip surface
mesh surfaces, drawing R2-168,	R2-344
UG-112	!Mouse system variable R2-542
message	MOVIE procedure R1-466
See also error handling	MPROVE procedure R1-468
error, setting level to report in "DC"	!Msg_Prefix system variable R2-542
functions R1-159	multidimensional arrays, using subscripts
for incomplete "DC" function	PG-82
R1-157	multiple
issuing error R1-460	array elements, reading records with
popup message PG-449	PG-181—PG-182, PG-185
suppressing compiler messages	plots UG-83
PG-29	statements, separating with
MESSAGE procedure R1-460, PG-251,	ampersand UG-35
PG-261-PG-262	multiplication operator (*) PG-32
Microsoft Word, inserting PV-WAVE plots	
into UG-A-45-UG-A-47	N
MIN function R1-462	/4
minimizing images UG-140	N ELEMENTS function R1-469,
minimum operator (<)	PG-270
examples PG-46	N PARAMS function R1-471, PG-269
operator precedence PG-32	N_TAGS function R1-472, PG-119
minimum value of array R1-462	NaN, not a number PG-263
mixed data types PG-35, PG-42,	natural
PG-281	exponential function R1-317
MOD operator	·
description of PG-49	logarithm R1-38
operator precedence PG-32	
MODIFYCT procedure R1-464, UG-330	

PV-WAVE Index XXXIII

NE operator	opening files
description of PG-50	DC (Data Connection) functions
operator precedence PG-33	PG-140
nearest neighbor method UG-170	LUNs PG-138
nested procedures, getting information on	OPEN procedures R1-480, PG-139
PG-403	when to open PG-140
nonblocking window	XDR PG-205
dialog box PG-453	OPENR procedure R1-480
popup message PG-449	OPENU procedure R1-480
non-printable characters	OPENW procedure R1-480
examples PG-23	OpenWindows GUI PG-479
specifying PG-22	operands, checking validity of PG-264
normal coordinate systems UG-60,	operating system
UG-61	accessing using SPAWN PG-297
normalized coordinates R2-512	communicating with from
not equal	PV-WAVE PG-291
See NE operator	UNIX, calling from PV-WAVE R1-77
NOT operator	operator precedence
description of PG-50	examples PG-33
example PG-43	table of PG-32
operator precedence PG-32	operators PG-47
number of elements in expression	addition (+) PG-45, PG-123
PG-270	AND PG-48
numeric	array concatenation, table of PG-8
constants PG-4	assignment (=) PG-44
errors	Boolean, table of PG-7
machine specifics PG-268	division PG-45
trapping PG-263	EQ PG-48
operators, table of PG-7	evaluating PG-42
	exponentiation (^) PG-46
	GE PG-49
	general description PG-31
object modules, callwave, compiling	grouping PG-44
PG-313	GT PG-49
octal characters, for representing	hierarchy of PG-32
non-printable characters PG-23	infix notation PG-2
offset parameter PG-216, PG-217	LE PG-49
ON_ERROR procedure	LT PG-49
description R1-474, PG-258	matrix multiplication (#) PG-46
examples PG-250	maximum (>) PG-46
table of values PG-259	minimum (<) PG-46
ON_IOERROR procedure	MOD PG-49
description R1-479, PG-258	multiplication PG-45
uses of PG-259	NE PG-50
OPEN procedures PG-139	NOT PG-50
,	numeric, table of PG-7
	, -

XXXIV PV-WAVE Index

PV-WAVE Index XXXV

keyword PG-74	wider than the physical width of the
number of non-keyword in routines	screen UG-A-87
R1-471	with X Windows UG-A-8, UG-A-85
number required in routines PG-236	plot
passing PG-75, PG-246	area, defining R2-67, R2-79
by reference PG-246	axis format R2-526
by value PG-246	combining images and contours
mechanism PG-111, PG-234,	UG-100
PG-246	data window UG-85
positional, definition of PG-235	
parent widget, manages child PG-484	inhibiting clipping of R2-546 line styles R2-502
parentheses operator() PG-44	
!Path system variable R2-550	margin for annotation R2-525,
•	R2-530, R2-534
patterns Sacrahusan fill nathawa	multiple plots on a page R2-545
See polygon fill, pattern	position of in window R2-547
PCL output UG-A-23—UG-A-27	positioning in display window R2-79
percent sign (%), meaning of PG-A-6,	region UG-85
PG-A-22	specifying
period symbol UG-35	margin around R2-515,
Pgflquo quota PG-287	R2-555
physical memory, need for PG-283	range of data to plot R2-526,
physical record, definition of PG-147	R2-530, R2-534, R2-556
!Pi system variable R2-552	symbol index R2-516, R2-547
PICT output UG-A-27—UG-A-30	PLOT procedure
pipes, UNIX, description of PG-310	2D graphs R1-497
pixel interleaving PG-190, PG-192	basic description of UG-61
pixels	date/time examples UG-244,
copying UG-A-72, UG-A-85	UG-246, UG-249
data PG-192	for tables UG-286-UG-287
easy way to ascertain exact value	PLOT_FIELD procedure R1-507
R2-327	PLOT_IO procedure UG-84-UG-85
hardware UG-A-96	2D graphs R1-497
inverting region of UG-A-95	PLOT OI procedure UG-61
number per centimeter on graphics	2D graphs R1-497
device R2-539	PLOT_OO procedure
palette color PG-192	2D graphs R1-497
reading back UG-142	example of UG-82
scalable UG-139	PLOTERR procedure R1-505
values, reading from an image	PLOTS procedure R1-510
R2-15	plotters, HPGL UG-A-13
pixmaps	plotting
animating R2-278, R2-316	See also annotation, axes, color,
creating with WINDOW procedure	ticks, image processing
UG-A-86	!P system variable R2-543, PG-27
examples UG-A-86	
champles Ou-A-00	2D array as 3D plot R2-197
	3D data UG-93

XXXVI PV-WAVE Index

3D orientation of text R2-521	logarithmic
3D to 2D transformations R2-549	axes R2-533
adding	scaling UG-84
axes UG-87	main title R2-523
bar charts UG-76	marker symbols UG-72
changing coordinate systems	multiple UG-83
UĞ-59	number of points to sum R2-547
clipping	overlaying contour plots and images
output R2-503	UG-100
window R2-543	overplotting R1-488, UG-56,
combination plots R2-104	UG-66
connecting symbols with lines	polar
R2-548	coordinates R2-515
contour plots R1-119, R2-506	plots UG-89
controlling	polygon filling R1-542, UG-74
placement of cursor R2-217	polygons R1-560
converting from 3D to 2D coordinates	position of graphics cursor R1-143,
UG-119	UG-90
coordinate systems UG-59,	region filling UG-74
UG-115	routines, summary of UG-56
creating menus R2-221	scaling XY UG-63-UG-64
date/time data UG-243—UG-253	smoothing contour plots UG-108
examples UG-243-UG-253	speeding up R2-547
in tables UG-267	surface shading parameters
defining a plotting symbol R2-235,	UG-132
UG-72	surfaces UG-112
display of images R2-212, R2-225	symbols R2-547, UG-72
enhanced contour plots UG-99	system variables UG-55
error bars R1-305	tables UG-286-UG-287
exchange of axes UG-126	with a date/time axis UG-284
exposing and hiding windows	thickness of lines R2-549
R2-363	three-dimensional graphics UG-115
filling contour plots UG-109	transformation matrices UG-116
flushing output buffer R1-294	user-defined symbols UG-73
font selection R2-507	vector fields from 3D arrays R2-240
ganging UG-83	window display UG-A-6
highlighting with POLYFILL UG-74	windows
histogram UG-70	creating R2-356
input from the cursor UG-90	deleting R2-275
irregular polygon fill R1-542	POINT_LUN procedure
keywords R2-497	description R1-519, PG-219
line thickness R2-522	example PG-197
linear log scaling R1-497	used to access VMS files PG-225
location of plot in window R2-548,	
UG-85	
00-00	

PV-WAVE Index XXXVII

pointer	hardware UG-A-18
function	pattern R2-506, R2-514
example PG-488	polygon lists
WtPointer R2-390	description of UG-187
into files R1-519	merging R1-557
polar	polygonal meshes
coordinates R2-515, UG-89	defining with MESH function
plots R2-515, UG-89	R1-457
POLY function R1-521	example of UG-204-UG-205
POLY_2D function R1-526, UG-103,	polygons
UG-168, UG-172	calculating area of R1-522
POLY_AREA function R1-522	filling
POLY_C_CONV function R1-524,	See polygon fill
R1-535, UG-192	generating R1-22, R1-571, UG-187
POLY_COUNT function R1-537,	manipulating R1-23, UG-192
UG-192	merging lists for rendering R1-557
POLY_DEV function R1-538, UG-194	plotting R1-560
POLY_FIT function R1-552, UG-72	querying subscripts of pixels inside
POLY_MERGE procedure R1-557,	R1-550
UG-192	rendering R1-23, R1-560, R2-28,
POLY_NORM function R1-559, UG-194	UG-176, UG-195
POLY_PLOT procedure R1-560,	table of UG-183
UG-195	returning
POLY_SPHERE procedure R1-568,	list of colors for R1-524,
UG-190	R1-535
POLY_SURF procedure R1-571,	
UG-189	number contained in list R1-537
POLY_TRANS function R1-573,	shading R1-564
UG-192, UG-194	polylines UG-121
POLYCONTOUR procedure	polynomial
description R1-529	curve fitting R1-552, R1-555
example of UG-110	functions, evaluating R1-521
filling closed contours R2-514,	warping of images R1-526, R1-574
UG-109	POLYSHADE function R1-564, UG-195
syntax UG-109	POLYWARP procedure R1-574,
POLYFILL procedure	UG-173
description R1-542	
example UG-74, UG-76	POPD procedure R1-578, PG-303 popup menu
fill pattern R2-506, R2-514	
POLYFILLV function R1-550	avoid using with text widget R2-462
POLYFITW function R1-555	creating PG-427
polygon fill	defining menu items PG-429
2D or 3D polygon R1-542	description R2-461
color R2-504	example PG-427
example UG-74	popup message PG-449, PG-451 popup widget PG-419, PG-449,
Champio Od-14	
	PG-453, PG-457, PG-459

XXXVIII PV-WAVE Index

portable data, XDR PG-205-PG-206	call stack PG-262
portrait orientation UG-A-36	mechanism PG-248
position, of plot R2-515, UG-85	checking for
positional parameters	number of elements R1-469,
checking for PG-269	PG-270
examples PG-235	parameters R1-422, R1-472,
with functions PG-238	PG-269
positioning	compiling
file pointer PG-219	automatically PG-69, PG-234,
images on the display UG-138	PG-239
text in PostScript UG-A-39	three methods of PG-239
PostScript output UG-A-31—UG-A-48	with .RUN and filename
PRINT procedure	PG-239
description R1-580, PG-156	with interactive mode PG-240
for tables with columns UG-285	creating PG-15
	with a text editor UG-22
for writing structures PG-116	definition of PG-11, PG-233
PRINTD procedure R1-582, PG-303	definition statement of PRO PG-76,
Printer Control Language output	PG-235
See PCL output	deleting from memory R1-241
printers supported by PV-WAVE UG-A-2	determining number of parameters
PRINTF procedure	PG-269
description R1-580, PG-156	
for writing structures PG-116	directory path for R2-550, PG-28
in relation to journaling UG-37	documentation of user-written rou-
printing	tines R1-264
files R1-580	error handling PG-250, PG-257-
graphics output UG-A-1	PG-258
LN03 printer R1-434	formal parameters PG-235
tables with column titles UG-285	getting a list of those compiled
to LJ-250 printer UG-A-20	PG-401
to PostScript printer UG-A-32	help with R1-410
values of date/time variables	I/O errors in R1-479
R1-282	information on nested PG-403
variables R1-580	keyword parameters PG-74
private colormaps UG-A-81	libraries of VMS PG-251
PRO statement	library UG-23
format of PG-76	maximum size of code PG-397
syntax PG-235	number of
Procedure Call statement PG-73	non-keyword parameters
procedures	R1-471, PG-269
action to take on error R1-474	parameters required PG-236
actual parameters PG-234	parameters PG-74, PG-234
automatic compiling, conditions for	passing mechanism PG-75,
PG-69	PG-246
call statement PG-76	positional PG-235
calling PG-11	

PV-WAVE Index XXXIX

program control PG-257, PG-273	programming
recursion PG-248	accessing large arrays PG-281
required components of PG-237	avoid IF statements PG-276
search path UG-20, UG-46	code size PG-397
stopping execution R2-138	commenting programs PG-52
syntax for	delaying program execution R2-274
calling PG-15	format strings PG-A-1—PG-A-2
creating PG-15	menus, creating R2-221, R2-359
user-written PG-69	minimize virtual memory allocation
various sources of PG-73	PG-288
process, spawning R2-126, PG-297,	running out of virtual memory
PG-298	PG-284
processes, concurrent R1-309	tips PG-498
processing images	and cautions PG-472
See image processing	for writing efficient code
product-moment correlation coefficient	PG-275
R1-133	on loops PG-281
PROFILE function R1-583	use correct type of constants
PROFILES procedure R1-585	PG-280
profiles, extracting from images R1-583	use optimized system routines
program	PG-280
calling from within PV-WAVE	use vector and array data oper-
PG-310	ations PG-278
code area full PG-242	use virtual memory efficiently
control routines PG-257, PG-273	PG-285
creating and running UG-22	using arrays efficiently PG-279
data area full PG-243	writing efficient programs
file search method UG-48	PG-275
files containing PV-WAVE proce-	prompt
dures UG-28	changing R2-552, UG-52
format of UG-21, UG-24	PV-WAVE prompt string UG-12
help with nested PG-403	PROMPT procedure R1-587
increasing speed of PG-276	!Prompt system variable R2-552
linking to PV-WAVE PG-310	PSEUDO procedure R1-588, UG-330
listings UG-28	Pseudo Color, 12-bit UG-A-94
main PV-WAVE UG-24	pseudo-color PG-189
maximum size of code PG-397	compared to true-color UG-146
pausing execution R2-274	images, PostScript UG-146
preparing and running UG-12	PUSHD procedure R1-590, PG-303
running as batch UG-19	PV-WAVE session
stopping R2-138	exiting R1-316
submitting to Standard Library	getting information about PG-395
PG-255	
. 4 200	recording R1-419
	restoring R2-33
	saving R2-56

XI PV-WAVE Index

$\boldsymbol{\alpha}$	range
u	setting default for axes R2-81
QMS QUIC output UG-A-48—UG-A-51 quadric animation, example of UG-207 QUERY_TABLE function combining multiple clauses UG-280 description R2-1, UG-7, UG-263 Distinct qualifier UG-272 examples PG-179, UG-265 features UG-270 Group By clause UG-273 In operator UG-280 passing variable parameters UG-279 rearranging a table UG-271 renaming columns UG-272 sorting with Order By clause UG-276 syntax UG-271 Where clause UG-277 quick.mk makefile PG-346 !Quiet system variable R2-553 QUIT procedure R2-11, UG-14 quitting PV-WAVE R1-316, R2-11, UG-13 quotas Pgflquo PG-287 Wsquo PG-287 quotation marks UG-35 quoted string format code PG-A-10, PG-A-19	setting detault for axes 112-01 subscript, for selecting a subarray PG-84 raster graphics, colors UG-A-92 raster images UG-135 rasterizer, Tektronix 4510 UG-A-61 ray tracing cone primitives R1-113 cylinder primitives R1-150 definition of UG-197 description of UG-175, UG-196 mesh primitives R1-457 RENDER function R2-28 sphere primitives R2-129 summary of routines R1-23 viewing demonstration programs UG-177 volume data R2-272 RDPIX procedure R2-15 READ procedure R2-17, PG-156, PG-160—PG-161 READF procedure description PG-156 example PG-176 with STR_TO_DT function PG-170 for fixed format PG-175 for row-oriented FORTRAN write PG-183 reading
0	ASCII files R2-17 reading
R	24-bit image file R1-195
!Radeg system variable R2-553 radians converting from degrees PG-27 converting to degrees R2-553, PG-27 radio button box widget PG-436 random file access PG-225 random number, uniformly distributed	8-bit image data R1-192, PG-189 ASCII files R1-161, R1-177, R2-17 binary files between different systems PG-205 binary input R2-17 byte data from an XDR file PG-206—PG-207 C-generated XDR data PG-207— PG-208
R2-13 RANDOMN function R2-12	cursor position R1-143, UG-90

PV-WAVE Index XII

RANDOMU function R2-13

data	record-oriented
date/time UG-228	binary files PG-149
files R2-17	data, transferring in VMS files
from a file PG-12	PG-150
from a word-processing applica-	records
tion PG-175	definition of PG-147
from multiple array elements	extracting fields from PG-37
PG-180	fixed length format (VMS) PG-217
DC_READ routines PG-A-8	PG-226
files, using C programs PG-196	length of PG-147
fixed-formatted ASCII files R1-161	multiple array elements PG-180,
freely-formatted ASCII files R1-177	PG-181, PG-182, PG-185
from magnetic tapes R2-193	recovering from errors PG-258
TAPRD PG-229	PV-WAVE session R2-33
from the display device UG-142	rectangular surfaces UG-189
images from the display R2-223	recursion PG-248
multiple array elements with FOR-	reference, parameter passing by
TRAN format string PG-185	PG-246
READF PG-175-PG-176, PG-181,	REFORM function R2-24
PG-183, PG-185	region of interest, selecting R1-232
for arrays PG-181	Regis output UG-A-54-UG-A-56
READU PG-195, PG-203	register, event handler PG-487
records with multiple array elements	REGRESS function R2-26
PG-180, PG-182, PG-185	regression, fit to data R2-26
TIFF files R1-198	relational operators
unformatted input R2-17	descriptions PG-48-PG-50
XDR files with READU procedure	evaluating with operands of mixed
PG-209	data types PG-42
READU procedure	table of PG-8, PG-41
binary files PG-195	using with arrays PG-42
description PG-154	Remote Procedure Call
reading binary input R2-17	See RPC
syntax PG-193	RENDER function
with Segmented keyword PG-203	color, defining for UG-198
XDR files PG-209	cone objects R1-113, UG-196
realize, widget hierarchy PG-484 REBIN function	cylinder objects R1-150, UG-197
decrease sampling with UG-98	default view UG-202
description R2-21	defining material properties of objects UG-200
resampling original image with UG-103	description R2-28, UG-196, UG-203
using to magnify or minify an image	example of UG-204-UG-218
UG-141	invoking UG-203
recalc flag, in !DT structure UG-226 record attributes of VMS files PG-226	lighting model UG-197
recording a PV-WAVE session R1-419	mesh objects R1-457, UG-197
10001ding a 1 V-VVAVE 56551011 R1-419	sphere objects R2-129, UG-197

xlii PV-WAVE Index

syntax UG-203 volume objects R2-272, UG-197	REWIND procedure R2-38, PG-229 RGB
rendering	color system UG-306
images, displaying UG-219	triplets PG-193
iso-surfaces UG-216	RGB TO HSV procedure R2-39
polygons R1-23, R1-560, UG-183,	right-handed coordinate system UG-116
UG-195	RMS files, reading images PG-217
process of UG-185	Roberts edge enhancement R2-45
ray-traced objects R2-28, UG-196	ROBERTS function R2-45, UG-160
shaded surfaces R1-564, R2-83,	root window
	relationship to visual class UG-A-90
R2-91	See shell window
volumes R1-24, R2-96, R2-267,	ROT function R2-48
UG-182–UG-183, UG-195	ROT Idiction R2-46 ROT_INT function R2-54
with standard techniques UG-195	
REPEAT statement PG-10, PG-77	ROTATE function
REPLICATE function	description R2-51
description R2-31	examples UG-138
description of PG-114	syntax UG-168
examples PG-195	rotating
reserved LUNs	arrays or images R2-51
description PG-140-PG-141	corresponding to surface R2-168
operating system dependencies	current color table R2-302, UG-320
PG-142	data UG-117
reserved words PG-25	images R2-48, R2-51, R2-54
reserving colors for other applications use	row-oriented
UG-A-84	ASCII data PG-146
resizing arrays or images R2-21	data, shown in figure PG-146
resource	FORTRAN write PG-183
data type of value PG-483	rows
definition of PG-483	in arrays PG-47
deriving name of PG-483	removing duplicate from a table
file PG-415, PG-482	UG-270
setting for Widget Toolbox PG-483	transposing with columns PG-47
resource file	RPC
how to use PG-464	description of PG-359
RESTORE procedure R2-33, UG-40-	example, CALL_UNIX PG-377—
UG-41	PG-387
RETALL procedure R2-34, PG-251	synchronization of processes
RETURN procedure R2-35, PG-68,	PG-360
PG-76, PG-236, PG-251	running
reversal of rows and columns PG-47	See executing
REVERSE function R2-36	run-time compilation of statements
reversing	PG-273
current color table R2-302, UG-320	
direction of vector R2-36	
reversion, format PG-167, PG-A-5	

PV-WAVE Index XIIII

S	description R2-439
	example PG-448
sampled images UG-135	multiple selection mode PG-447
SAVE procedure R2-56, UG-40	single selection mode PG-447
saving	searching VMS libraries PG-252
a PV-WAVE session R1-419, R2-56	SEC_TO_DT function R2-58, UG-233 seconds, converting to date/time vari-
compiled procedures R1-104	ables R2-58
TIFF data PG-191	semicolon after @ symbol UG-20
scalable pixels UG-102, UG-139	sensitivity, of widgets PG-470
scalars	servers
combining with subscript array,	C program as PG-380
ranges PG-91	closing connections UG-A-72
converting to date/time variables	definition of PG-359
R2-238	example program(test_server.c)
definition of PG-4, PG-24	PG-369
in relation to arrays PG-290	in X Window systems UG-A-69
subscripting PG-84	linking with PV-WAVE PG-361
scale unit cube into viewing area R2-57	UNIX REPLY function PG-375
SCALE3D procedure R2-57, UG-123	using PV-WAVE as PG-371-
scaling	PG-373, PG-392–PG-394
corresponding to surface R2-168	session
data UG-117	See PV-WAVE session
images R2-225	SET_PLOT procedure
input images with BYTSCL function	description R2-66
UG-154	SET_SCREEN procedure R2-67
logarithmic UG-84	SET_SHADING procedure R2-70,
plots UG-63-UG-64	ŪG-132
three-dimensional R2-57	SET_SYMBOL procedure R2-74,
Y axis with YNozero UG-64	PG-296
screen pixels, assigning color UG-A-78	SET_VIEW3D procedure R2-77,
scroll bars	UG-194
on drawing area PG-439	SET_VIEWPORT procedure R2-79
ScrollBarcallbackparameters (Motif)	SET_XY procedure R2-81
PG-C-5	SETBUF function
scrollbar callback parameters (OLIT)	example PG-311
PG-C-8	with child program PG-310
scrolling list PG-446	SETDEMO procedure R2-62
text widget PG-443	setenv command

Xliv PV-WAVE Index

used to view

scrolling list

image R2-326

text R2-351

callback PG-447 creating PG-446

for WAVE_PATH UG-47

for WAVE_STARTUP UG-49

with WAVE_DEVICE UG-45

SETENV procedure R2-64, PG-293

SETLOG procedure R2-65, PG-295

SETUP_KEYS procedure R2-75

description R2-83, UG-131 examples UG-133 SHADE_SURF_IRR procedure R2-91 SHADE_VOLUME procedure description R2-96, UG-190 example of UG-206 shadding constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters Gr shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80-UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-484 unmanaging PG-484 unmanaging PG-484 realizing PG-484 unmanaging PG-484 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107	SHADE_SURF procedure	Fast Fourier Transform R1-328
examples UG-133 SHADE_SURF_IRR procedure R2-91 SHADE_VOLUME procedure description R2-96, UG-190 example of UG-206 shading constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-484 unmanaging PG-484 unmanaging PG-484 unmanaging PG-484 show a widget PG-489 show a widget PG-489 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shrifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 histogram equalization R1-383 Lee filter R1-424 simultaneous equations, solution of R2-179 SINDGEN function R2-109 sine, hyperbolic R2-111 sine function R2-109 sine, hyperbolic R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-179 SINH function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-17, R2-329 sliders and text input field PG-437 creating PG-437 description R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		Hanning filter R1-378
SHADE_SURF_IRR procedure SHADE_VOLUME procedure description R2-96, UG-190 example of UG-206 shading constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80-UG-A-81 shared library, io.so R2-62 shared colormaps UG-160 shell processes PG-299-PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-488 managing PG-484 unmanaging PG-484 realizing PG-484 unmanaging PG-484 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 Lee filter R1-424 simultaneous equations, solution of R2-177 SIN function R2-109 sine, hyperbolic R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-117 SIN function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-117 SIN function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 SINET function R2-112 SIXEL output UG-A-58-UG-A-61 SIZE function R2-179 SINH function R2-112 SIXE function R2-179 SIXE function R2-114, PG-272 examples PG-272 sta	examples UG-133	histogram equalization R1-383
SHADE_VOLUME procedure description R2-96, UG-190 example of UG-206 shading constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80-UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299-PG-300 shell processes PG-299-PG-300 shell window creating PG-484 umanaging PG-484 UB-182 With initial layout PG-414 destroying PG-488 managing PG-484 UB-183 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 simultaneous equations, solution of R2-1109 SINDGEN function R2-109 sine, hyperbolic R2-111 sine function R2-109 sine, hyperbolic R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-179 SINF function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 SIZE function description R2-112 SIXEL output UG-A-58-UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-158 smoothing plane, defining R2-253 volumes R2-117, R2-329 slicers and text input field PG-437 creating PG-437 creating PG-437 smoothing SIMMatnetion R2-119 SIXEL output UG-A-58-UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure PG-346 SLICE_VOL function R2-117, UG-179, UG-158 simultaneous equations single-precision floating output PG-444 shown in figure P3-176 SIXEL output UG-A-58-UG-A-61 SIZE fun		Lee filter R1-424
description R2-96, UG-190 example of UG-206 shading constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters or shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80-UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-484 realizing PG-484 realizing PG-484 realizing PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function constant intensity R2-72 sIN function R2-109 sine, hyperbolic R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-179 SINF function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-117 SIX function R2-112 SIXEL output UG-A-58-UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-272 examples PG-270 SINF function R2-117 SIX function R2-179 SIXEL output UG-A-58-UG-A-61 SIZE function R2-117, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-272 examples PG-272 size of arrays,		simultaneous equations, solution of
example of UG-206 shading constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80-UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299-PG-300 shell window creating PG-416, PG-482 with initial alyout PG-414 destroying PG-484 realizing PG-484 realizing PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function Constant intensity R2-72 since function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-179 SINH function R2-112 SIXEL output UG-A-58-UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-345 shown in figure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-345 shown in figure R2-117, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-272 examples PG-273 size function R2-177 SVD function R2-179 SINH function R2-117 SVD function R2-117 SVD function R2-117 SVD function R2-114 single step command UG-27, UG-30 single-precision floating single recision floating s		R2-177
shading constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80-UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299-PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-484 realizing PG-484 realizing PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SINDGEN function R2-110 sine function R2-112 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-179 SINH function R2-112 SIXEL output UG-A-58-UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 stable of type codes PG-272 size of arrays, determining R2-114, PG-272 examples PG-272 SiXIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158	·	SIN function R2-109
constant intensity R2-72 contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-488 managing PG-484 realizing PG-484 srealizing PG-484 srealizing PG-484 shell, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 shifting elements of arrays R2-99 shifting images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 single step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirit adding to surface R2-345 shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 Sixe formation R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirit adding to surface R2-345 shown in figure PG-446 SLICE VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 usingle step command UG-27, UG-30 single-precision floating-point data shown in figure PG-146 SIZE function Curve R2-177 SVD function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function Curve R2-177 SVD function R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirit	•	SINDGEN function R2-111
contour plots R1-529 examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80-UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299-PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-484 realizing PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-112 SIXEL output UG-A-58-UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-345 shown in figure R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SNDOTH function R2-119, UG-158 smoothing		sine function R2-109
examples UG-133 Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-488 realizing PG-484 unmanaging PG-484 shell_L, UNIX environment variable PG-293 Shift function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-119 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-119 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-177 SVD function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-114, PG-2170 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-3		sine, hyperbolic R2-112
Gouraud R2-71 light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 shared library, io.so R2-62 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-488 managing PG-484 realizing PG-484 realizing PG-484 shell L, UNIX environment variable PG-293 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 single-precision floating-point data shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-179 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-270 examples PG-272 size of arrays, determining R2-114, PG-272 size of arrays, determining R2-114, PG-272 examples PG-272 siz		
light source UG-131 methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharly mindow creating PG-416, PG-482 with initial layout PG-414 destroying PG-488 managing PG-484 unmanaging PG-484 unmanaging PG-484 unmanaging PG-484 Unmanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shirting elements of arrays R2-99 shirting images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 shown in figure PG-146 singular value decomposition curve R2-177 SVD function R2-179 SINH function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 table of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure PG-146 Singular value decomposition curve R2-177 SVD function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure PG-436 SICE—VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-437 Size of arrays R2-99 shirt function R2-115, PG-229 skirt adding to surface R2-345 shown in figure PG-446 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-345 size of arrays, determining R2-117, PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-115, PG-229 skirt		
methods UG-131 setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 realizing PG-484 unmanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 svD function R2-117 SVD function R2-111 slXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158		
setting parameters UG-132 setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-484 realizing PG-484 unmanaging PG-484 shell, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 shifting elements of arrays R2-99 shirikning images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 curve R2-177 SVD function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-272 size of arrays, determining R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-279 size of arrays pG-272 size of arrays pg-272 size of	g .	
setting parameters for shading R2-70 surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 unmanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SINH function R2-112 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 sharel library, io.so R2-62 sharepening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-484 realizing PG-484 realizing PG-484 summanaging PG-484 summanaging PG-484 summanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 size of arrays, determining R2-114, PG-272 SiZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 size of arrays, determining R2-114, PG-272 SiZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 size of arrays, determining R2-114, PG-272 SiZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 shiller adding to surface R2-345 shown in figure R2-345 shown	setting parameters for shading	
surfaces R1-564, R2-83, R2-91, R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 realizing PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SIXEL output UG-A-58—UG-A-61 SIZE function description R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 size of arrays, determining R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-272 examples PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 shirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 slicing plane, defining R2-253 volumes R2-117, R2-329 slicing plane, defining R2-253 volumes R2-117, R2-329 slici		÷ · · · · · · · ·
R2-345, UG-131 setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 realizing PG-484 shift in unmanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 shifting elements of arrays R2-99 shifting images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SIZE function description R2-114, PG-272 examples PG-272 stable of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 sikirt adding to surface R2-345 shown in figure R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
setting parameters R2-79 volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 shared colormaps UG-A-81 shared library, io.so R2-62 shared colormaps UG-A-81 stable of type codes PG-272 size of arrays, determining R2-114, PG-270 size of arrays, determining R2-114, PG-272 size of arrays, determining		
volumes R2-96 shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 shared colormaps UG-A-81 shared library, io.so R2-62 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 Slicing plane, defining R2-253 volumes R2-117, R2-329 sliders SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SMOOTH function R2-119, UG-158 smoothing		
shared colormaps UG-A-80—UG-A-81 shared library, io.so R2-62 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 stable of type codes PG-272 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
shared library, io.so R2-62 sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 unmanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 size of arrays, determining R2-114, PG-270 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
sharpening, of images UG-160 shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 unmanaging PG-484 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders SHIFT function R2-99 shifting elements of arrays R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SKIPF procedure R2-115, PG-229 SKIPF procedure R2-115, PG-229 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-345 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
shell processes PG-299—PG-300 shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 realizing PG-484 realizing PG-484 shell window Skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders And text input field PG-437 creating PG-437 description R2-414 show3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SKIPF procedure R2-115, PG-229 skirt adding to surface R2-345 shown in figure R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
shell window creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 unmanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 skirt adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
creating PG-416, PG-482 with initial layout PG-414 destroying PG-468 managing PG-484 realizing PG-484 unmanaging PG-484 unmanaging PG-484 SHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 adding to surface R2-345 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing	•	
with initial layout PG-414 shown in figure R2-346 destroying PG-468 UG-193 realizing PG-484 UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders SHELL, UNIX environment variable PG-293 sliders SHIFT function R2-99 shifting elements of arrays R2-99 creating PG-437 show a widget PG-469 description R2-414 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 Shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-158 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, UG-158 shown in figure R2-346 SLICE_VOL function R2-117, UG-179, U		
destroying PG-468 managing PG-484 realizing PG-484 unmanaging PG-484 sHELL, UNIX environment variable PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SLICE_VOL function R2-117, UG-179, UG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
managing PG-484 realizing PG-484 unmanaging PG-484 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 MG-193 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		
realizing PG-484 unmanaging PG-484 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 slicing plane, defining R2-253 volumes R2-117, R2-329 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		-
unmanaging PG-484 plane, defining R2-253 SHELL, UNIX environment variable PG-293 sliders SHIFT function R2-99 and text input field PG-437 shifting elements of arrays R2-99 creating PG-437 show a widget PG-469 description R2-414 SHOW3 procedure R2-104, UG-129 example PG-438 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 SMOOTH function R2-119, UG-158 smoothing		
SHELL, UNIX environment variable PG-293 SHIFT function R2-99 Shifting elements of arrays R2-99 Show a widget PG-469 SHOW3 procedure R2-104, UG-129 Shrinking images R1-115, R2-21, R2-36 SIGMA function Computing standard deviation of array R2-107 SIGMA function Computing Standard deviation of array R2-107 SIGMA function SIGMA function Computing Standard deviation of array R2-107 SIGMA function SIGMA STANDARD SIGMA S		
PG-293 SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 sliders and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing	unmanaging PG-484	
SHIFT function R2-99 shifting elements of arrays R2-99 show a widget PG-469 SHOW3 procedure R2-104, UG-129 shrinking images R1-115, R2-21, R2-36 SIGMA function computing standard deviation of array R2-107 and text input field PG-437 creating PG-437 description R2-414 example PG-438 orientation R2-415 using PG-437 SMOOTH function R2-119, UG-158 smoothing		·
shifting elements of arrays R2-99 creating PG-437 show a widget PG-469 description R2-414 SHOW3 procedure R2-104, UG-129 example PG-438 shrinking images R1-115, R2-21, R2-36 orientation R2-415 SIGMA function using PG-437 computing standard deviation of array R2-107 SMOOTH function R2-119, UG-158 smoothing		
show a widget PG-469 description R2-414 SHOW3 procedure R2-104, UG-129 example PG-438 shrinking images R1-115, R2-21, R2-36 orientation R2-415 SIGMA function using PG-437 computing standard deviation of array R2-107 SMOOTH function R2-119, UG-158 smoothing		
SHOW3 procedure R2-104, UG-129 example PG-438 shrinking images R1-115, R2-21, R2-36 orientation R2-415 SIGMA function using PG-437 computing standard deviation of array R2-107 SMOOTH function R2-119, UG-158 smoothing		
shrinking images R1-115, R2-21, R2-36 orientation R2-415 SIGMA function using PG-437 computing standard deviation of array R2-107 SMOOTH function R2-119, UG-158 smoothing		
SIGMA function using PG-437 computing standard deviation of SMOOTH function R2-119, UG-158 array R2-107 smoothing	SHOW3 procedure R2-104, UG-129	
computing standard deviation of SMOOTH function R2-119, UG-158 array R2-107 smoothing		
array R2-107 smoothing	— · — · ·	•
	description R2-107	boxcar R2-119
	signal processing	
convolution R1-125 contours, using Spline keyword		- • • • • • • • • • • • • • • • • • • •
	creating filters R1-250	UG-108
	creating filters R1-250	UG-108

PV-WAVE Index XIV

mean UG-159	spheres
median UG-159	defining with SPHERE function
of images UG-158-UG-159	R2-129, UG-197
Sobel edge enhancement R2-122	generating R1-568
SOBEL function R2-122, UG-160	gridding R1-373
software fonts	spherical
index of R2-544	gridding UG-192
See fonts	surfaces UG-190
solids, shading R1-564	SPLINE function R2-132
SORT function R2-125, UG-290	splines
sorting	cubic R2-132
array contents R2-125	interpolation of contour plots
SORT function UG-290	R2-519
tables, using QUERY TABLE	SQL, description of UG-264
R2-1, PG-179-PG-180, UG-276	SQRT function R2-134
spaces in statements PG-52	square root, calculating R2-134
sparse data R1-368	stacking plots UG-83
spatial transformation of images	standard deviation, calculating R2-136
R1-526, R1-574	standard error output
SPAWN procedure	LUNs PG-142
accessing C programs from	reserved LUNs PG-140
PV-WAVE PG-312	standard input
avoiding shell processes PG-300	LUNs PG-141
calling without arguments PG-298	reserved LUNs PG-140
capturing output PG-301	Standard Library
communicating with a child process	location of PG-16, PG-255, UG-23
PG-310	submitting programs PG-255
description R2-126	suggestions for writing routines
example of PG-298, PG-312	PG-256
interapplication communication	standard output
PG-309	LUNs PG-141
non-interactive use PG-299	reserved LUNs PG-140
syntax PG-297	starting PV-WAVE
when to use PG-307	description UG-12
spawning a process R2-126	executing a command file UG-13
special characters, list of UG-33	from an external program PG-314
special effects	interactive UG-12
created with color UG-328	startup file
using Z-buffer UG-A-99	for UNIX UG-49
with color UG-A-94	for VMS UG-50
SPHERE function	statements
description of R2-129	assignment PG-9, PG-53
example of UG-207	blocks of PG-10, PG-60
	can change data types PG-55
	CASE PG-10, PG-62
	COMMON block PG-12, PG-63

xlvi PV-WAVE Index

compiling and executing with	STRING function
EXECUTE function PG-273	description R2-144
components of PG-52	example PG-37, PG-124
examples of assignment PG-54	formatting data PG-187
executing one at a time R1-313	syntax PG-124
FOR PG-9	with
function call PG-11	BYTE function PG-199
function definition PG-11	single byte argument PG-125
GOTO PG-10, PG-70	structures PG-118
groups of PG-60	string processing PG-121
IF PG-9, PG-70, PG-71	compressing white space R2-140
labels PG-52	converting
list of 12 types PG-51	lower to upper case R2-167
procedure	upper to lower case R2-150
calls PG-11, PG-73	extracting substrings R2-154
definition PG-11, PG-76	inserting substrings R2-157
REPEAT PG-10, PG-77	locating substrings R2-156
spaces in PG-52	removing
tabs in PG-52	blanks R2-140
types of PG-9, PG-51	excess white space R2-141
WHILE PG-10, PG-78	leading and trailing blanks
statically linked programs PG-333	R2-162
STDEV function	string variables
computing standard deviation	binary transfer of PG-197
R2-136	determining number of characters to
description R2-136	transfer PG-197
stdio facility PG-310	passing into QUERY_TABLE
STOP procedure R2-138	UG-279
stopping PV-WAVE UG-13	strings
storing image data PG-189	array arguments PG-133
STR TO DT function	arrays, creating R2-111, R2-139
description R2-159, UG-229,	changing case PG-127
UG-231	concatenating PG-123
example PG-171, UG-232	constants PG-4, PG-21
templates UG-231	definition of PG-121
STRARR function R2-139	determining length of PG-122
STRCOMPRESS function R2-140	examples of string constants PG-21
stream mode files, VMS PG-226	extracting substrings PG-131
STRETCH procedure R2-142, UG-155,	formatting PG-122
UG-330	FORTRAN and C formats PG-166
stretching the current color table	importing with free format PG-160
R2-302, UG-320	initializing to a known length
string data	PG-199
a basic type PG-3	input/output with structures PG-117
shown in figure PG-146	inserting substrings PG-131
-	

PV-WAVE Index XIVII

length issues with structures	STRUCTREF function PG-104
PG-118	description R2-164
locating substrings PG-131	structure references
non-string arguments PG-133	examples PG-109
obtaining length of PG-130	nesting of PG-108
operations supported PG-122	syntax PG-109
operators, table of PG-8	structures
reading with Format keyword	See also unnamed structures
PG-166	advanced usage PG-119
removing white space PG-122,	arrays PG-112, PG-114
PG-128	creating unnamed PG-105
STRCOMPRESS function PG-122	data type PG-3
STRING function PG-122	date/time R1-204, R1-211, R1-290
string operations supported PG-121	UG-225
STRLEN function PG-122	defining PG-102
STRLOWCASE function PG-122	definition of PG-101
STRTRIM function PG-122	deleting PG-102
STRUPCASE function PG-127	expressions PG-39
substrings PG-131	formatted input/output PG-116
transferring as binary data PG-197	getting information about R1-410,
working with PG-121	PG-110, PG-402
writing to a file PG-164	importing data into PG-161
strip chart	in relation to tables UG-287—
controlling the pace R2-339	UG-288
displaying R2-336	input and output PG-116
STRLEN function	introduction to PG-101
description R2-148	listing references to R2-164
determining string length PG-122	number of tags in R1-472
example PG-130	N TAGS function PG-119
output PG-197	parameter passing PG-111
syntax PG-130	PRINT and PRINTF PG-116
STRLOWCASE function	processing with tag numbers
converting to lower case PG-122	PG-119
description R2-150	reference to a field PG-108
example PG-127	references to R2-164
syntax PG-127	removing definitions from memory
STRMESSAGE function R2-152	R1-242
STRMID function R2-154, PG-132	REPLICATE function PG-114
STRPOS function R2-156, PG-131	scope of PG-105
example PG-131	storing into array fields PG-112—
STRPUT procedure R2-157, PG-132	PG-113
STRTRIM function R2-162	string
examples PG-128	input/output PG-117
removing white space PG-122	length issues PG-118
syntax PG-128	structure references PG-109
•	subscripted references PG-108

xlviii PV-WAVE Index

Style field of IX, IY, !Z UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting in to strings R2-156, PG-131 subtraction operator (-) in a file R2-87 volumetric R2-307, R2-329 surface plot adding skirt R2-345 angle of X rotation R2-497 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with color of bottom R2-499 combining with rate color of bottom R2-499 combining with rate color of bottom R2-497 changing interactively R2-344 color of bottom R2-497 changing interactively R2-344 color of bottom R2-497 changing interactively R2-344 color of bottom R2-499 combining with color of bottom R2-499 combining with color R2-499 combining with color R2-499 combining with color R2-499 color of bottom R2-499 combining with color R2-499 color of bottom R2-497 displayed color of bottom R2-497 angle of Z rotation R2-499 combining with color R2-499 color indexing R2-104 color of bottom R2-499 combining vith color R2-499 and situation R2-172 displayed as R	unformatted input/output PG-117 using INFO PG-110 STRUPCASE function R2-166, PG-127 Style field of !X, !Y, !Z UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89 PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84-PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-51 with matrices PG-82 multidimensional arrays PG-57 with matrices PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 insertings R2-156, PG-131 subtraction operator (-) examples PG-45	tag names R2-188	SunOS Version 4, error handling
using INFO PG-110 STRUPCASE function R2-166, PG-127 Style field of !X, !Y, !Z UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-86— using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) See overlaying surface data in a file R2-87 volumetric R2-307, R2-329 surface plot adding skirt R2-345 angle of X rotation R2-497 changing interactively R2-344 color of bottom R2-497 angle of Z rotation R2-497 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-497 angle of X rotation R2-497 angle of Z rotation R2-497 angle of Z rotation R2-497 angle of Z rotation R2-497 angle of X rotation R2-499 and ring viting N2-172 delaphorumants of R2-172 display 3D array as R2-168 displaying data	using INFO PG-110 STRUPCASE function R2-166, PG-127 Style field of IX, IY, IZ UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89 PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53 using arrays as PG-57 using arrays as PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 See overlaying surface data in a file R2-87 volumetric R2-307, R2-329 surface plot adding skir R2-345 angle of X rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-497 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with mage and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with contour plot R2-517 placement of Z axis R2-533 rendering data usering array as R2-168 displaying data as R2-342 draw a skirt on R2-151 horizontal lines only R2-509 lower surface only R2-509 lower surface only R2-509 range PG-84 shorizontal lines	TAG_NAMES function PG-119	PG-268
using INFO PG-110 See overlaying Style field of !X, !Y, !Z UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89 PG-90 notation for columns and rows PG-81 fiber of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) See overlaying surface data in a file R2-87 volumetric R2-307, R2-329 surface plot adding skirt R2-345 angle of X rotation R2-497 changing interactively R2-499 combining with eractively R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-509 upper surface only R2-509 upper surface only R2-533 rendering of shaded R1-564, R2-93, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	using INFO PG-110 See overlaying StPUPCASE function R2-166, PG-127 StPyle field of IX, IY, IZ UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89 PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 See overlaying surface data in a file R2-87 volumetric R2-307, R2-329 surface plot adding skirt R2-345 angle of X rotation R2-497 changing interactively R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with image and contour plots R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-497 angle of X rotation R2-497 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with image and contour plots R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with image and contour plots R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with image and contour plots R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with image and contour plots R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 lower surface only R2-509 lower surface only R2-509 l	unformatted input/output PG-117	superimposing
STRUPCASE function R2-166, PG-127 Style field of !X, !Y, !Z UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-95 with SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting instrings R2-156, PG-131 subtraction operator (-) surface data in a file R2-87 volumetric R2-307, R2-329 surface plot adding skirt R2-345 angle of X rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-197 as swirtace plot adding skirt R2-345 angle of X rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with colour plot R2-172 display 3D reverded display 3D reverded insering inte	STRUPCASE function R2-166, PG-127 Style field of IX, IY, IZ UG-64 style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89—PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays as PG-57 with matrices PG-82 multidimensional arrays PG-95 subsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 subration R2-634, R2-334, R2-344 color of bottom R2-497 adding skirt R2-345 angle of Z rotation R2-497 adding skirt R2-345 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-497 changing interactively R2-344 color of b	using INFO PG-110	See overlaying
style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 subarrays, structure of PG-86 subarrays, structure of PG-86 subarrays, structure of PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) volumetric R2-307, R2-329 surface plot adding skirt R2-345 surface plot adding skirt R2-345 angle of X rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-156 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower surface only R2-508 ucher Ge-89 combining with image and contour plots R2-104 color of bottom R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combining with image and contour plots R2-104 color of bottom R2-499 combinin	style, of axes R2-526, R2-531, R2-534, R2-557 subarrays, structure of PG-86 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-11, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-155— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 volumetric R2-307, R2-345 angle of X rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-508 lower surface only R2-508 size field to the color of bottom R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-509 upper surface only R2-509 upper surface only R2-509 upper surface only R2-508 lower surface only R2-508 surface plot sadding skirt R2-345 sngle of X rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-		surface data
style, of axes R2-526, R2-531, R2-534, R2-537 surface plot subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89—PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-51 substring arrays as PG-57 with matrices PG-82 multidimensional arrays PG-84 structures PG-108 substrings R2-156, PG-131 from strings R2-154 inserting into strings R2-156, PG-131 substraction operator (–) style, of axes R2-526, R2-531, R2-534, surface plot adding skir R2-345 angle of X rotation R2-497 changing interactively R2-495 angle of Z rotation R2-518 horizon pa-497 changing i	style, of axes R2-526, R2-531, R2-534, R2-557 surface plot subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89—PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-66, PG-84 4 types PG-86—VG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-15, UG-277 substrings extracting PG-131 from strings R2-156, PG-131 subtraction operator (-) examples PG-45	Style field of !X, !Y, !Z UG-64	in a file R2-87
subarrays, structure of PG-86 submarrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 using arrays as PG-57 using arrays as PG-58 structures PG-108 subsetting data SQRT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings R2-156, PG-131 from strings R2-156, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) subtraction of PG-86 ading skirt R2-345 angle of X rotation R2-497 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower surface only R2-508 lower surface only R2-508 upper surface only R2-509 upper surface only R2-508 lower su	R2-557 subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subscriting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 sumples of X rotation R2-497 angle of X rotation R2-497 angle of X rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-503 easy way to change appearance R2-342, R2-344 example of C rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-508 lower surface only R2-508 sakirt on R2-497 displayang data as R2-342 draw a skirt on R2-497 displayang data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-5		volumetric R2-307, R2-329
subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-97 with matrices PG-84 structures PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-156, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) adding skirt R2-345 angle of X rotation R2-497 changing interaction R2-497 changing interaction R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 upper surface only R2-509 u	subarrays, structure of PG-86 submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-15, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 adding skirt R2-345 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower surface only R2-508 lower surface only R2-508 lower surface only R2-508 askirt on R2-499 combining with image and contour plots R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-499 combining with image and contour plots R2-108 display 3D array as R2-168 display 3D a	-	surface plot
submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting intot strings R2-156, PG-131 subtraction operator (-) angle of X rotation R2-497 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 display3D array as R2-168 display3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display3D array as R2-168 display3D array as R2-168 display3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344, example of combining with contour plot R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display3D array as R2-168 display3D array as PG-37 splay array as PG-50 P2-179 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 saving the 3D to 2D transformation R2-517 Saving the Ag-2-342 Saving the Ag-2-342 Sa	submatrices PG-98 subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 angle of X rotation R2-497 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-508 solicity of the proper surface only R2-509 upper surface only R2-508 solicity of the proper surface only R2-509 upper surface only R2-509 upper surface only R2-508 solicity of the proper surface only R2-509 upper surface only R	subarrays, structure of PG-86	
*operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-57 with matrices PG-82 multidimensional arrays PG-91 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) * operator PG-85 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-509 uppe	subscripts * operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-92 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 angle of Z rotation R2-497 changing interactively R2-344 color of bottom R2-497 changing interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower surface only R2-508 lower surface only R2-508 existing a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower surface only R2-508 reference syntax PG-82 supper surface only R2-508 lower surface only R2-508 lover s		
* operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting interactively R2-344 color of bottom R2-499 combining with interactively R2-498 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 low	* operator PG-85 combining arrays with PG-89— PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 subscript arrays PG-53—PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 Subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 cobaning interactively R2-344 color of bottom R2-499 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower surface only R2-508 lower surface only R2-508 lower surface only R2-508 reference syntax PG-82 super surface only R2-508 lower		angle of Z rotation R2-497
combining arrays with PG-89—PG-90 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw 4 types PG-84—PG-85 able of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-154 locating in strings R2-156, PG-131 subtraction operator (–) color of bottom R2-499 combining with image and contour plots R2-172 display 3D array as R2-198 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only	combining arrays with PG-89—PG-90 combining with image and contour notation for columns and rows PG-81 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw A types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 using arrays as PG-57 using arrays as PG-57 using elements in arrays PG-82 multidimensional arrays PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45	•	changing interactively R2-344
PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 subscript arrays PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) combining with image and contour plots R2-104 curve fitting to R2-117 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-509 easy way to change appearance a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-509 reference syntax PG-91 sakirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-509 reference syntax PG-91 reference syntax PG-82 upper surface only R2-509 upper surface only R2-509 upper surface only R2-509 reference syntax PG-91 reference syntax PG-92 upper surface only R2-509 reference syntax PG-91 reference syntax PG-92 upper surface only R2-509 reference syntax PG-91 reference syntax PG-92 reference syntax PG-82 reference syntax PG-82 reference syntax PG-82 reference syntax PG-95 reference syntax PG-82 reference syntax PG-95 reference syntax PG-91 reference syntax PG-95 reference syntax PG-95 reference syn	PG-90 notation for columns and rows PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 combining with image and contour plots R2-104 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-508 lower surface only R2-508 reasy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data uG-117 saving the 3D to 2D transformation R2-517 R2-345 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		
PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-52 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting in to R2-154 subtraction operator (–) curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-509 upper surface only R2-508 lower surf	PG-81 of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-508 lower surface only R2-508 reference syntax PG-91 a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 upper surface only R2-508 upper surface only R2-509 upper surface only R2-508 lower surface only R2-509 upper surface only R2-508 R2-342, R2-344 example of combining with contour R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 SORT function UG-290 Shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		combining with image and contour
of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 subscript arrays PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting in strings R2-156, PG-131 subtraction operator (–) curve fitting to R2-172 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-508 lower su	of pixels inside polygon region R1-550 ranges PG-56, PG-84 A types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (–) examples PG-45 of pixels inside polygon region display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-508 lower surface only R2-508 lower surface only R2-508 lower surface only R2-508 as kirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower su	notation for columns and rows	plots R2-104
of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 subscript arrays PG-57 using arrays as PG-57 using arrays as PG-57 using arrays as PG-57 using arrays as PG-57 using arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	of pixels inside polygon region R1-550 ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 using arrays as PG-57 using arrays as PG-57 using arrays as PG-57 usind arrays PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 display 3D array as R2-168 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2		curve fitting to R2-172
ranges PG-56, PG-84 4 types PG-86—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-57 using arrays as PG-57 using arrays as PG-57 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 subtraction operator (-) displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 lower surface only R2-508 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 Shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 subtraction operator (-) examples PG-45 displaying data as R2-342 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-508 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		
ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 using arrays as PG-57 using arrays as PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 subtraction operator (-) data 4 types PG-84	ranges PG-56, PG-84 4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 SUBSETTING data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 subtraction operator (—) examples PG-45 draw a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-523 easy way to change appearance R2-342, R2-344 R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 Solution R2-564, R2-345 shown with and without a skirt R2-346 supering of haded R1-564, R2-345 shown with and without a		
4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-15, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-138, R2-138 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, subtraction operator (-) UG-112	4 types PG-84—PG-85 table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-57 using arrays as PG-57 using arrays as PG-57 matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 subtraction operator (-) examples PG-45 a skirt on R2-518 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-509 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 Placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		
table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 using arrays as PG-57 matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	table of PG-86 using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 using arrays as PG-57 matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (-) examples PG-45 horizontal lines only R2-508 lower surface only R2-509 upper surface only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 R2-348, R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		a skirt on R2-518
using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-154 locating in strings R2-156, PG-131 subtraction operator (-) lower surface only R2-509 upper surface only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	using asterisk UG-36 reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) examples PG-45 lower surface only R2-509 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-346 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		horizontal lines only R2-508
reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-154 locating in strings R2-156, PG-131 subtraction operator (–) upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	reference syntax PG-82 storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 upper surface only R2-523 easy way to change appearance R2-342, R2-344 example of combining with contour R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		
storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 subtraction operator (–) easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	storing elements in arrays PG-91 subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) examples PG-45 easy way to change appearance R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		
subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 SUBsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 subtraction operator (–) R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	subscript arrays PG-53—PG-57 using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 SUBsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-342, R2-344 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		easy way to change appearance
using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 subtraction operator (-) example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	using arrays as PG-57 with matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 subtraction operator (-) examples PG-45 example of combining with contour plot R2-517 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 subtraction operator (-) examples PG-45 examples of combining with contour plot R2-513 subtraction of Shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 Subtraction gate with contours R2-517 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		
with plot R2-517 matrices PG-82 placement of Z axis R2-533 multidimensional arrays PG-82 rendering of shaded R1-564, scalars PG-84 structures PG-108 rotating data UG-117 subsetting data SORT function UG-290 shaded R1-564, R2-517 SORT function UG-290 shaded R1-564, R2-345 Where clause UG-277 shaded R1-564, R2-345 substrings extracting PG-131 R2-346 substrings extracting PG-131 R2-517, UG-125 from strings R2-154 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, subtraction operator (-)	with plot R2-517 matrices PG-82 placement of Z axis R2-533 multidimensional arrays PG-82 rendering of shaded R1-564, scalars PG-84 R2-83, R2-91 structures PG-108 rotating data UG-117 subsetting data UG-117 subsetting SORT function UG-290 shaded R1-564, R2-345 Where clause UG-277 shaded R1-564, R2-345 substrings extracting PG-131 R2-346 substrings extracting PG-131 R2-517, UG-125 from strings R2-154 superimposed with contours extracting into strings R2-157—R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 with R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 saving the 3D to 2D transformation R2-517 subtraction gata UG-117 saving the 3D to 2D transformation R2-517 subtraction gata UG-117 saving the 3D to 2D transformation R2-517 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,	using arrays as PG-57	
matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (–) matrices PG-82 placement of Z axis R2-533 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 Subtraction of R2-345 Superimposed with contours R2-346 Superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure dure UG-125, UG-127 description R2-168, UG-93, UG-112	matrices PG-82 multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 subtraction operator (-) examples PG-45 multidimensional arrays PG-82 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of R2-345 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of R2-345 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of R2-345 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of R2-345 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of R2-345 subtraction of Saving the 3D to 2D transformation R2-517 subtraction of R2-345 subtra		•
multidimensional arrays PG-82 scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 locating in strings R2-156, PG-131 subtraction operator (–) multidimensional arrays PG-82 rendering of shaded R1-564, R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	multidimensional arrays PG-82 rendering of shaded R1-564, scalars PG-84 structures PG-108 rotating data UG-117 subsetting saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 where clause UG-277 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours extracting PG-131 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-346 superimposed with contours R2-517, UG-125 superimposed with contours R2-517, UG-125 superimposed with CONTOUR procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, subtraction operator (-) examples PG-45 examples UG-112, UG-114,		placement of Z axis R2-533
scalars PG-84 structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-156, PG-131 subtraction operator (–) R2-83, R2-91 rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	scalars PG-84 structures PG-108 rotating data UG-117 subsetting saving the 3D to 2D transformation R2-517 SORT function UG-290 shaded R1-564, R2-345 Where clause UG-277 shown with and without a skirt R2-346 substrings extracting PG-131 superimposed with contours R2-517, UG-125 from strings R2-154 superimposed with contours R2-517, UG-125 inserting into strings R2-157— combining with CONTOUR procedure Inserting in strings R2-156, PG-131 subtraction operator (-) examples PG-45 examples UG-112, UG-114,		rendering of shaded R1-564,
structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 subtraction operator (–) rotating data UG-117 saving the 3D to 2D transformation R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	structures PG-108 subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 rotating data UG-117 saving the 3D to 2D transformation R2-517 R2-517 Shaded R1-564, R2-345 superimposed with contours R2-346 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		
subsetting saving the 3D to 2D transformation R2-517 SORT function UG-290 shaded R1-564, R2-345 Where clause UG-277 shown with and without a skirt R2-346 substrings extracting PG-131 superimposed with contours R2-517, UG-125 from strings R2-154 superimposed with contours R2-517, UG-125 inserting into strings R2-157— combining with CONTOUR procedure UG-125, UG-127 locating in strings R2-156, PG-131 description R2-168, UG-93, Subtraction operator (-)	subsetting data SORT function UG-290 Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) examples PG-45 sORT function UG-290 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		rotating data UG-117
data R2-517 SORT function UG-290 shaded R1-564, R2-345 Where clause UG-277 shown with and without a skirt R2-346 substrings extracting PG-131 R2-517, UG-125 from strings R2-154 SURFACE procedure inserting into strings R2-157— combining with CONTOUR procedure Incenting in strings R2-156, PG-131 subtraction operator (-) R2-517 Shaded R1-564, R2-345 shown with and without a skirt R2-346 SURFACE procedure combining with CONTOUR procedure Incention R2-168, UG-93, UG-112	data SORT function UG-290 Where clause UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) examples PG-45 R2-517 shaded R1-564, R2-345 shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,		saving the 3D to 2D transformation
Where clause UG-277 tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157 R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) shown with and without a skirt R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	Where clause UG-277 shown with and without a skirt R2-346 substrings extracting PG-131 R2-517, UG-125 from strings R2-154 SURFACE procedure inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) examples PG-45 shown with and without a skirt R2-346 superimposed with contours R2-346 superimposed with contours R2-517, UG-125 superimposed with contours R2-158, PG-125 superimposed with contours R2-517, UG-125 superimposed with contours R2-1517, UG-125 superimposed with contours R2-158, PG-125 superimposed with contours R2-1517, UG-125 superimposed with contours R2-1517, UG-125 superimposed with contours R2-158, PG-125 superimposed with contours R2-1517, UG-125 superimposed with CONTOUR procedure under UG-125, UG-127 superimposed under UG-125, UG-	3	R2-517
tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	tables R2-1, UG-277 substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 R2-346 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,	SORT function UG-290	
substrings superimposed with contours extracting PG-131 R2-517, UG-125 from strings R2-154 SURFACE procedure inserting into strings R2-157— combining with CONTOUR proce- R2-158, PG-122, PG-131 dure UG-125, UG-127 locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (–) UG-112	substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) examples PG-45 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,	Where clause UG-277	shown with and without a skirt
substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112	substrings extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 superimposed with contours R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,	tables R2-1, UG-277	R2-346
extracting PG-131 R2-517, UG-125 from strings R2-154 SURFACE procedure inserting into strings R2-157— combining with CONTOUR proce- R2-158, PG-122, PG-131 dure UG-125, UG-127 locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (–) UG-112	extracting PG-131 from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 R2-517, UG-125 SURFACE procedure combining with CONTOUR procedure loure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,	•	superimposed with contours
from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (–) SURFACE procedure combining with CONTOUR procedure dure UG-125, UG-127 description R2-168, UG-93, UG-112	from strings R2-154 inserting into strings R2-157— R2-158, PG-122, PG-131 locating in strings R2-156, PG-131 subtraction operator (-) examples PG-45 SURFACE procedure combining with CONTOUR procedure dure UG-125, UG-127 description R2-168, UG-93, UG-112 examples UG-112, UG-114,	•	
inserting into strings R2-157— combining with CONTOUR proce- R2-158, PG-122, PG-131 dure UG-125, UG-127 locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (–) UG-112	inserting into strings R2-157— combining with CONTOUR proce- R2-158, PG-122, PG-131 dure UG-125, UG-127 locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (-) uG-112 examples PG-45 examples UG-112, UG-114,		SURFACE procedure
R2-158, PG-122, PG-131 dure UG-125, UG-127 locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (–) UG-112	R2-158, PG-122, PG-131 dure UG-125, UG-127 locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (–) UG-112 examples PG-45 examples UG-112, UG-114,		combining with CONTOUR proce-
locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (–) UG-112	locating in strings R2-156, PG-131 description R2-168, UG-93, subtraction operator (–) UG-112 examples PG-45 examples UG-112, UG-114,		
subtraction operator (–) UG-112	subtraction operator (–) UG-112 examples PG-45 examples UG-112, UG-114,	locating in strings R2-156, PG-131	description R2-168, UG-93,
	examples PG-45 examples UG-112, UG-114,		UG-112
Champios 1 a 10			examples UG-112, UG-114,
	operator precoderies 1 & sz	operator precedence PG-32	UG-119

PV-WAVE Index XIIX

list of keywords OG-113	getting information about PG-403
transformation matrix R2-169	listing their values PG-403
example UG-119	passing of PG-246
SURFACE FIT function R2-172	relationship to keywords UG-24,
SURFR procedure R2-174	UG-57
suspending PV-WAVE	summary of PG-27
on a UNIX system UG-14	systems, window
on a VMS system UG-14	See window systems
SVBKSB procedure R2-177	SYSTIME function R2-184
SVD procedure R2-179	31311ME IUNCUON R2-164
SVDFIT function	
description R2-181	<i>T</i>
example of basis function for	T3D procedure UG-116
R1-137	data transformation UG-116,
swap area, increasing PG-285	UG-118
symbol	description R2-185
displaying in a volume R2-263	examples UG-119
plotting R2-547	setting up data for viewing UG-194
VMS, defining R2-74	table functions
VMS, returnable R1-361	description of UG-263
symbols	overview UG-264
connecting with lines R2-516,	table widget, creating PG-462
UG-72	tables
marker UG-72, UG-73	creating with BUILD_TABLE
plotting R2-516	function R1-63, UG-266—
size of plotting R2-520	UG-270
user-defined R2-516, UG-73	date/time data UG-281
VMS, deleting R1-238	determining unique values R2-230
SYSGEN parameters	GUI tool R2-472
VirtualPageCount PG-286	in relation to structures UG-287—
Wsmax PG-286	
system	UG-288
color tables UG-320	plotting UG-286—UG-287
date, creating variable with R2-199	date/time axis UG-284
functions, using for efficient program-	printing with column titles UG-285
ming PG-280	rearranging with QUERY_TABLE
time, creating variable with R2-184,	UG-271
R2-199	renaming columns UG-272
system limits and compiler PG-241	sorting R2-1
•	columns in descending order
system variables	UG-276
adding R1-236	with QUERY_TABLE function
creating R1-236	PG-179-PG-180
definition of PG-26	subsetting R2-1
description of PG-5, UG-24	with Where clause UG-277
exclamation point UG-34	
for tick label formats UG-82	

unique elements of R2-230	text
viewing with INFO procedure	3D orientation of R2-521
UG-268	See also annotation, fonts
tabs in statements PG-52	alignment R2-497
Tag Image File Format	angle of R2-513
See TIFF	centered R2-497
TAG_NAMES function R2-188, PG-119 tags	changing case R2-150, R2-166, PG-127
definition of PG-101	character size R2-501, R2-518
names, of structures R2-188	color of R2-504
number of R1-472	concatenation PG-123
reference to structure PG-108	converted from byte data PG-125
TAN function R2-190	converting parameters to R2-168,
	PG-124
tangent R2-190	editor programs UG-22
hyperbolic R2-191	extracting PG-131
TANH function R2-191	inserting PG-131
tape, magnetic	locating substrings PG-131
accessing under VMS PG-229	non-string arguments to string rou-
mounting a tape drive (VMS)	tines PG-133
PG-230	obtaining length of strings R2-148,
reading block of image data PG-231	PG-130
REWIND procedure PG-229	orientation of R2-513
rewinding R2-38	positioning commands, for Post-
SKIPF procedure PG-229	Script UG-A-39
skipping	removing white space PG-128
backward on a tape PG-231	size of R2-501, R2-518
forward on the tape (VMS)	string operations supported PG-121
PG-230	suppressing clipping R2-511
TAPRO procedure PG-229	vector-drawn UG-291
TAPWRT procedure PG-229 WEOF procedure PG-229	viewed in scrolling window R2-349,
writing to PG-229	R2-351
TAPRD procedure R2-193, PG-229	width of R2-522
TAPWRT procedure R2-194, PG-229	working with PG-121
TEK_COLOR procedure R2-195,	writing to the display R2-490
UG-326, UG-330	text widget
Tektronix 4115 device, mimicking colors	and popup menus R2-462
UG-330	creating PG-443
Tektronix 4510 rasterizer UG-A-61—	description R2-481
UG-A-64	multi-line PG-445
Tektronix terminal	scroll bars PG-443
color table R2-195	single line
Tektronix terminal output UG-A-65—	editable PG-444
UG-A-68	read-only PG-443
terminating, execution of a command file	
UG-20	

thickness	format of R2-527
of axis line R2-558	intervals between R2-528,
of characters R2-543	R2-532, R2-535
THREED procedure R2-197	intervals, setting number of
three-dimensional	R2-560
coordinate systems	linestyle of R2-507, R2-525,
converting to two-dimensional	R2-529, UG-78
coordinate systems UG-119	number of minor marks
user-defined UG-121	R2-525, R2-530, R2-534,
data, See three-dimensional data	R2-556, UG-78
graphics	producing non-linear marks
combining with images	UG-78
UG-129-UG-130	suppressing minor marks
drawing UG-115	R2-556
rotating data UG-117	values of R2-526, R2-530,
scaling data UG-117	R2-534
shaded surfaces R1-564,	TIFF
R2-83, R2-91, R2-185	conformance levels PG-192
SHOW3 procedure R2-104	data
translating data UG-117	
gridding R1-322, R1-367, UG-191	saving in TIFF format PG-191
transformations	writing image data to a file
of text UG-292	R1-221, R1-223 writing with DC_WRITE_TIFF
with two-dimensional proce-	PG-191
dures UG-124	files
volumes UG-190	
three-dimensional data	compression of PG-192
contouring R1-119, UG-94	reading R1-198 reading a file in R1-198
displayed as surface R2-168	
MOVIE procedure R1-466	writing image data to a file R1-221, R1-223
plotting UG-93	time
viewing as iso-surface R2-307	current R2-184
threshold	
dithering UG-148	elapsed between dates R1-277
value of an iso-surface R2-312	formats, STR_TO_DT function
thresholding of images UG-152	UG-231
tick	timer, adding PG-489
intervals, setting number of UG-66	tips, programming PG-472, PG-498
label format UG-81–UG-82	title
marks	See also annotation
	adding to two-dimensional plots
annotation of R2-527, R2-531, R2-535	UG-64
· - • • •	of axes R2-529, R2-532, R2-535,
controlling length of R2-522, R2-550, UG-78	R2-560
extending away from the plot	of plot R2-523, R2-550
R2-522, UG-78	setting size of R2-501, R2-543
nz-322, 0G-70	TODAY function R2-199, UG-259

lii PV-WAVE Index

toggle button, menu PG-430	transmission component of color, for
tool box R2-485, PG-433-PG-435	RENDER function UG-199
top-level window	transparency, in images R1-543
See shell window	TRANSPOSE function R2-204, PG-47
TOTAL function	transposing
description R2-200	arrays or images R2-51, R2-204
example of PG-280	rows and columns PG-47
use of PG-47	TRED2 procedure R2-207
total, of array elements R2-200	TRIDAG procedure R2-208
TQLI procedure R2-201	TRNLOG function R2-209, PG-295
traceback information PG-262	true
transferring	definition for IF statement PG-71
binary data PG-154	definitions for different data types
binary string data PG-197	PG-71
complex data, with XDR routines	representation of PG-42
PG-210	true-color
	compared to pseudo-color UG-146
data with C or FORTRAN formats	images
	definition UG-147
PG-165	generating R1-406
READU, WRITEU PG-194	truth table
XDR files PG-206	for AND operator PG-43
date/time data PG-168	
images in server UG-A-82	for OR operator PG-43
transformation	for XOR operator PG-43
3D volumes R2-270	TV procedure
geometric UG-167-UG-168	default coordinates for UG-59
gray level UG-152	description R2-212, UG-135
matrix, See transformation matrices	TVCRS procedure R2-217
saving R2-517	description UG-136, UG-143
transformation matrices R2-521	example UG-143
3D points R1-573	TVLCT procedure
4-by-4 R2-270, R2-549	description R2-219, UG-136,
description UG-116	UG-311
rotating data UG-117	examples UG-325
scaling data UG-117	uses of UG-145
Transform keyword for RENDER	TVMENU function PG-473
function UG-202	TVRD function
translating data UG-117	description R2-223, UG-136,
with POLY_TRANS function	UG-142
R1-573	examples UG-142
with SURFACE procedure UG-119	TVSCL procedure
with T3D procedure UG-118	description R2-225, UG-136
translation table	examples UG-153
bypassing UG-A-71, UG-A-85	24-bit image data
color UG-308	writing data to a file R1-221
translucency, in images R1-561, R2-267	how stored PG-189
and a control of the	

24-bit color, using color to differentiate	routines for transferring PG-156
data sets UG-A-93	VMS FORTRAN generated, example
two-dimensional	of reading PG-202
coordinate systems, converting from	unformatted I/O
three-dimensional coordinate sys-	description PG-159
tems UG-119	with associated variables PG-212
data, summary of plotting routines	with structures PG-117
UG-56	unformatted string variables, I/O PG-160
gridding R1-319, R1-364	uniformly distributed random numbers
2D gridding UG-191	R2-13
type conversion	UNIQUE function R2-230, UG-7,
byte R1-68	UG-264
complex R1-109	unique values, determining R2-230
double-precision R1-267	UNIX
floating-point R1-340	
functions	avoiding shells with SPAWN
examples PG-37	PG-300
overflow conditions PG-36	calling from PV-WAVE R1-77
	commands from within PV-WAVE
syntax PG-38	R2-17, R2-33, R2-56, R2-64,
table of PG-36	R2-114, R2-126, PG-293—
integer R1-338	PG-297
longword integer R1-442	CALL_UNIX PG-359
scaled byte R1-73	Control-\ UG-15
string R2-144	compiling for LINKNLOAD PG-322,
type of variable, determining R2-114,	PG-326, PG-328
PG-272	description of files in PG-223
types	environment variable
See data types	adding R2-64
	changing R2-64
<i>)</i>	inspecting R1-295
	translation R1-356
undefined results	WAVE_DEVICE UG-45
Infinity PG-263	WAVE_DIR UG-45
NaN PG-263	WAVE_PATH UG-47
undefined variables, checking for	WAVE_STARTUP UG-49
PG-270	environment, description of PG-292
unexpected colors UG-A-96	FORTRAN programs
unformatted data	in relation to ASSOC function
advantages and disadvantages of	PG-218
PG-151	writing to PV-WAVE PG-200
FORTRAN generated, reading in	linking
UNIX environment PG-199	C program to PV-WAVE
problems reading PG-205	PG-347
reading with associated variable	FORTRAN applications to
method PG-212	PV-WAVE PG-348
otilod 1 d E1E	offset parameter PG-216
	5.100. paramotor 1 G 210

liv PV-WAVE Index

structure tag names R2-188	view setup
system	3D view R2-77
See system variables	centering data R1-86
types of PG-24	defining R2-253
valid names PG-26	description of UG-194
virtual memory PG-284	summary of routines R1-23
VAX/VMS Extended Metafile System	VIEWER procedure R2-253, UG-179,
UG-A-13	UG-194
vector	viewport, defining R2-67, R2-79
cross product R1-142	virtual memory
definition of PG-4, PG-24	description PG-283
fields, plotting R1-510, R2-244	in relation to
from 3D arrays R2-240	PV-WAVE PG-283
of strings R2-149, R2-151, R2-167	swap area PG-285
reversing R2-36	minimizing use of PG-288
solution, improving R1-468	under UNIX PG-285
subscripts PG-86	variable assignments PG-284
vector graphics colors UG-311,	VMS PG-286
UG-A-92	VirtualPageCount parameter PG-286
VECTOR_FIELD3 procedure R2-240,	visual class(es)
UG-195	for X windows UG-A-77
vector-drawn text	not inherited by PV-WAVE UG-A-90
See text	visual data analysis, importance of color
vectors	UG-310
BUILD_TABLE function UG-269	VMS
building tables from R1-63	access mode PG-225
definition of PG-24	accessing magnetic tape PG-229
deriving unique elements from	binary CGM output UG-A-13
R2-230	binary files PG-149
using as subscripts to other arrays	calling PV-WAVE R1-429
PG-88	command interpreter PG-299
VEL procedure R2-244	commands from within PV-WAVE
VELOVECT procedure R2-248	PG-295-PG-297
Ventura Publisher, inserting PV-WAVE	environment variables PG-295—
plots into UG-A-47—UG-A-48	PG-296, UG-46
version number of PV-WAVE R2-553,	error handling PG-268
PG-29	file
!Version system variable R2-553	access PG-225
vertex lists	attributes PG-227
description of UG-187	organization PG-224
merging R1-557	formal parameters for procedures
video memory, of workstation UG-A-79	and functions PG-236
View Control window R2-259	FORTRAN programs, writing to a file
View Orientation window R2-260	PG-202
	libraries PG-251, PG-253

lvi PV-WAVE Index

linking a C program to PV-WAVE	slicing
PG-349	example of UG-210
logical names, deleting R1-240	with SLICE_VOL function
mounting a tape drive PG-230	R2-117
offset parameter PG-217	transforming R2-270
record attributes PG-226	volumetric surface data R2-307, R2-329
reserved LUNs PG-142	VT graphics terminals UG-A-54
RMS files, reading images PG-217	3 1
searching libraries PG-252	W
sending output file to printer or plotter	V V
UG-A-6	w amond roply function PG-364
specific information on data files	w_cmpnd_reply function PG-364,
PG-224	PG-368
	w_get_par function PG-366
stream mode files PG-226	w_listen function PG-365
symbols	w_send_reply function PG-364, PG-367
defining R2-74	w_simpl_reply function PG-364
deleting R1-238	w_smpl_reply function PG-367
return value R1-361	WAIT procedure R2-274
transferring record-oriented data	waiting, in programs R2-274
PG-150	warping of images R1-526, R1-574,
variable length format PG-226	UG-168
virtual memory PG-286	WAVE Widgets
working set size PG-286	application example PG-473
VOL_MARKER procedure R2-263,	arranging PG-420
UG-195	basic steps in creating PG-413
VOL_PAD function R2-266, UG-192	combining with Widget Toolbox
VOL_REND function R2-267, UG-195	PG-480
VOL_TRANS function R2-270, UG-193	creating PG-416
VOLUME function	event loop PG-471
description of R2-272	getting values PG-466
example of UG-211-UG-218	hiding PG-469
volumes	initializing PG-414
defining R2-253	introduction PG-410
with VOLUME function	list of PG-418
R2-272, UG-197	location of PG-411
displaying markers in R2-263	passing user-defined data PG-467
generating UG-187	sensitivity PG-470
manipulating R1-24, UG-192	setting
padding R2-266	colors PG-463
rendering R1-24, R2-28, R2-240,	fonts PG-464
R2-263, R2-267, UG-183-	values PG-466
UG-195	showing PG-469
shaded R2-96	submitting to Users' Library PG-411
	widget hierarchy PG-416
	WAVE DEVICE UG-44-UG-45

PV-WAVE Index Ivii

WAVE_DIR UG-45-UG-46	WglsoSurfTool procedure
WAVE_PATH UG-46UG-47	description R2-307
WAVE_STARTUP UG-49-UG-50	event handling R2-312
wavecmd routine	WgMovieTool procedure
description of PG-313	benefits of R2-319
examples PG-315, PG-316,	description R2-315
PG-318	event handling R2-320
when to use PG-307	WgSimageTool procedure
with FORTRAN programs PG-315	description R2-323
waveinit routine PG-313-PG-314	event handling R2-327
wavestartup file UG-49	WgSliceTool procedure
waveterm routine PG-313, PG-315	description R2-329
wavevars function	event handling R2-334
accessing data in PV-WAVE	shown in figure R2-332
PG-350	slices three-dimensional data
description of PG-321, PG-335	R2-329
examples PG-354-PG-358	WgStripTool procedure
parameters PG-350	description R2-336
syntax PG-321, PG-350	event handling R2-340
WDELETE procedure R2-275, UG-A-98	WgSurfaceTool procedure
WEOF procedure R2-276, PG-229	benefits of R2-344
WgAnimateTool procedure	description R2-342
description R2-277	event handling R2-347
event handling R2-281	WgTextTool procedure
WgCbarTool procedure	description R2-349
description R2-284	event handling R2-352
event handling R2-287	Where clause UG-277
stores colors in common block	WHERE function R2-354, PG-55,
R2-288	PG-58, PG-277, UG-A-23
WgCeditTool procedure	WHILE statement PG-10, PG-78
benefits of R2-291	white space
compared to WgCtTool procedure	compressing R2-141
R2-291	removing R2-140, R2-162
description R2-290	removing from strings PG-122,
event handling R2-299	PG-128
stores colors in common block	widget
R2-293	class PG-482
utility widgets, shown in figure	definition of PG-410, PG-482
R2-298	destroy PG-485
WgCtTool procedure	hierarchy
compared to WgCeditTool R2-302	See widget hierarchy
description R2-301	ID (handle) PG-418, PG-482
event handling R2-305	input focus of PG-484
stores colors in common block	managing PG-484
R2-303	mapped with an X window PG-484 realizing PG-484

Iviii PV-WAVE Index

resources PG-483	window systems
resources, retrieving R2-376	backing store UG-A-7
sensitivity PG-470	common features UG-A-6
setting resources PG-465	defining with WAVE_DEVICE
unmanaging PG-484	UG-44-UG-45
widget hierarchy	features of those supported UG-A-6
close PG-485	retained UG-A-7
definition PG-416	table of UG-A-2
displaying PG-471	X Windows UG-53, UG-A-69
Widget Toolbox	WMENU function R2-359
-	word-processing application, reading
adding	data from PG-175
callbacks PG-485	words, reserved PG-25
event handlers R2-367,	working set
PG-487	-
timers PG-490	description PG-283 quota PG-287
combining with WAVE Widgets	·
PG-480	VMS PG-286
creating widgets PG-482	write mask
cursors PG-D1-PG-D5	interacts with selected graphics
destroying widgets PG-484	function UG-A-95
displaying widgets PG-484	using to create special effects
example program PG-495	UG-328, UG-A-94
include files PG-495	WRITEU procedure
initializing PG-481	description R2-361, PG-154
managing widgets PG-484	example PG-195
resources PG-483	syntax PG-193
running an application PG-494	transferring strings with PG-197
window	writing
creating R2-356	24-bit image data to a file R1-221
current R2-362	8-bit image data to a file R1-219
damage repair UG-A-7	ASCII free format data to a file
deleting by ID number UG-A-98	R1-203, R1-212
exposing and hiding R2-363	data, output formats PG-164
looks dark and unreadable	date/time data UG-255
UG-A-82	integer data, using format reversion
margin around plot R2-525,	PG-A-5-PG-A-6
R2-530, R2-534	strings to a file PG-164
menus, creating R2-221, R2-359	TIFF image data to a file R1-223
positioning plot in R2-67, R2-79	to a data file, basic steps PG-12
selecting R2-362	to end of file on tape (VMS) PG-229
specifying plot coordinates R2-515	to tape (VMS) PG-229
turning into icon R2-363	using
	UNIX FORTRAN programs
window index, used with drawing area	PG-200
PG-439	VMS FORTRAN PG-202
WINDOW procedure R2-356	WRITEU PG-195
	WHILE FG-195

WSET procedure R2-362
WSHOW procedure R2-363
Wsquo quota PG-287
WtAddCallback function R2-364,
PG-485
WtAddHandler function R2-367,
PG-487
WtClose function R2-369, PG-485
WtCreate function R2-371, PG-480,
PG-482
WtCursor function R2-373
WtGet function R2-376
WtInit function R2-380, R2-432, PG-480
WtInput function R2-382
WtList function R2-385
WtLoop function R2-387, PG-494
WtMainLoop function R2-389
WtPointer function R2-390
WtSet function R2-392, PG-480
WtTable function R2-395
WtTimer function R2-402, PG-489
WtWorkProc unction R2-404 wvsetup (WVSETUP.COM) file
methods of executing UG-12
WwButtonBox function R2-406, PG-434
WwCommand function R2-410, PG-459
WwControlsBox function R2-414, PG-437
WwDialog function R2-419, PG-453
WwDrawing function R2-422, PG-439
WwFileSelection function R2-426,
PG-456
WwGetValue function PG-466
description R2-430
WwInit function PG-413—PG-414
WwLayout function R2-434, PG-420
WwList function R2-439, PG-446
WwLoop function R2-443, PG-414
WwMainWindow function R2-444,
PG-416
WwMenuBar function R2-447, PG-426
WwMenuItem function R2-451
WwMessage function R2-453, PG-449
WwOptionMenu function R2-457,
PG-428

WwPopupMenu function R2-461, PG-427
WwRadioBox function R2-464, PG-436
WwSetValue function R2-468, PG-466
WwTable function R2-472
WwText function R2-481, PG-443
WwToolBox function R2-485, PG-435



X server closing connection UG-A-72 connecting to PG-414 !X system variable, fields of R2-554-R2-560 X Window System \$DISPLAY environment variable UG-A-69 clients UG-A-69 color translation UG-A-84 damage repair UG-A-7 DECW\$DISPLAY logical UG-A-69 DEVICE procedure UG-A-71 graphics function codes UG-A-75 IDs UG-A-97 keywords UG-A-71 overview UG-A-69 pixmaps UG-A-85-UG-A-86 private colormaps UG-A-82 providing a GUI for application UG-A-70 resources PG-464 servers UG-A-69 shared colormaps UG-A-81 using with PV-WAVE UG-53 visual classes UG-A-77 X11 socket R2-61 .Xdefaults file PG-465 **XDR** data allowed I/O routines PG-205 byte variables PG-206 differences from normal I/O PG-205

IX PV-WAVE Index

```
conventions for reading and
            writing PG-210
         creating with C programs
            PG-208
         description of PG-205
         opening PG-205
         reading, byte data PG-206,
            PG-207
         reading, READU procedure
            PG-209
     routines
         for transferring complex data
            PG-210
         table of PG-210
 Xlib PG-406
 XOR operator
     description of PG-50
     example UG-A-95
     operator precedence PG-33
     truth table PG-43
 Xt Intrinsics PG-406, PG-414, PG-481
 Xt TimeOut function PG-489
 XY plots
     examples UG-62
     scaling axes UG-63-UG-64
 XYOUTS procedure R2-490, UG-69,
     UG-121
Y
 Y axis, scaling with YNozero UG-64
 !Y system variable, fields of R2-561
Z
 !Z system variable, fields of R2-561
 Z-buffer
     output
         description UG-A-99
         DEVICE procedure UG-A-99
         keywords UG-A-99
     using to create special effects
        R1-543, UG-A-99
```

string variables PG-206 transferring PG-206

files

```
ZOOM procedure R2-493, UG-142
zooming
data in strip chart R2-337
images R2-493, UG-140
in 3D window R1-86
of reference cube R2-309, R2-331
ZROOTS procedure R2-495
```

PV-WAVE Index Ixi

Ixii PV-WAVE Index