Parsec Command Language (PCL)

Main Table of Contents

Plotting

AXES

_xaxis is a user defined vector of type integer or double.   It sets the scale for the x-axis of the plot, and the values at which a labeled tick is to appear on the x axis of the plot box. The first value of the vector is at the left edge, the last value is at the right edge, and the first is allowed to be larger than the last if the direction of increase is desired reversed.   The rest of the elements can appear in any order, as long as they lie in value between the first and the last.   If they fall outside the range, then the ticks will fall outside the rectangle, and the plot will appear strange.   To allow PCL to set the ticks automatically (which is the default), set *_xaxis* to a vector of length 1, or a scalar.   This is typically accomplished with the command *_xaxis=0.*

_yaxis is similar to *_xaxis*, but controls the ticks and the scale on the y - axis of the graph.

Labeling the Plot

Every plot needs labeling to make it comprehensible.   PCL provides for a main title, a legend on each axis, and an automatic date and signature in the corner.
_ti is a character string variable that contains the main title to the graph.   It appears centered above the top line of the plot box. Its default is the null string (no characters), so that there is no title unless explicitly set by the user.   For example, to title the plot "Intensity vs. Wavelength'', use the command
_*ti*= '*Intensity vs. Wavelength*'

_xt and ***_yt*** are similar to _*ti* except that they provide the axis labeling for the x and y axes respectively.
If the user is generating many plots over a period of time, it is convenient to have a record on the graph with the users name (or initials) and the date. ***_autodate*** and ***_autoname*** provide that.   _*autodate* is an integer flag.   If it is set to 1 (the default), then the date will appear in the lower corner of the graph. If it is set to 0, the date will be suppressed. _*autoname* is a string variable with a null default.   If the user inserts his/her name (eg _*autoname*='*J. Doe*'), then the name will appear with the date in the lower corner.   These two variables are typically set in the defaults.irt file.

Colors
PCL will support color plots to the screen. The usual approach to setting the colors is through the two system parameters ***_fg*** and ***_bg*.** Each is set to an integer which represents a color on the screen. *_fg* controls the foreground color (the colors of the lines), while *_bg* controls the background color.

The basic colors provided are those as defined for an EGA or VGA screen. color	number	name
black	0	*black*
blue	1	*blue*
green	2	*green*
cyan	3	*cyan*
red	4	*red*
magenta	5	*magenta*
brown	6	*brown*
light gray	7	*lightgray*
dark gray	8	*darkgray*
light blue	9	*lightblue*
light green	10	*lightgreen*
light cyan	11	*lightcyan*
light red	12	*lightred*
light	13	*lightmagenta*
yellow	14	*yellow*
white	15	*white*

For example, *_fg=4* before a call to plot will cause the box and the lines to be plotted in red instead of the default white. PCL defines sixteen parameters with names from the third column to serve as memory aids. For example, there is a parameter called brown, with an integer value of 6, so a command *_fg=brown* will cause the plot lines and labels to be brown. BUT, watch out! Do not inadvertently redefine these variables if you intend to use them.
The other way to plot in color is to use the third calling parameter of the PLOT type routines (PLOT, OPLOT, LINE, POINT). The third calling parameter is an integer, either scalar or vector. If scalar, it plots the points in the color associated with the value of the integer, while the box and background are set by *_fg* and *_bg* as usual. If it is a vector, then the colors will be matched to the ordered pairs one by one, so that multi-colored plots may be easily made. If the color vector is shorter than x and y, then the extra x and y points will be plotted in the color of the last element of the color array. Double precision color inputs are OK, they are simply truncated and used as integers.

```
;plots sixteen radians of the sine function color coded.
x=indices(159)/10.
y=sin(x)
c=fix(x)
plot,x,y,c
end
```

Plotters
The Edit and command windows can output their text to any printer through the windows interface by use of the menu command *print*. The output of graphics is a little more complicated. The simplest output is the use of the print command when the graphics window is selected. This will perform a bitmap output to the printer. Some printers are not well configured to handle the bitmap and may fail. In such a case, the first thing to try is to use the *save* menu feature, which will store the graphic as a device independent bitmap. This can be read into more sophisticated graphics programs, such as Paintbrush, which may then provide the needed interface to your printer.
The other drawback of printing the bitmaps is the relatively coarse resolution. IRT provides a mechanism for creating disk files of graphics commands for HP and PostScript printers. These disk files will provide full laser printer quality output, but will not be identical to the bitmap seen on the screen.

The printers supported are:

plotter	name	variable name
Bitmaps	'pc'	pc
Post Script	'ps'	ps
Hewlett Packard LaserJet	'hp'	hp

The input to the *plotter* command takes a character string variable. PCL initializes parameters by name, so that *printer,'ps'* and *printer,ps* will generate the same result unless the user inadvertently redefines the parameter *ps* .
PCL will save the graphics commands for ps and hp in a file of the name stored in the print file parameter *_pf* . A call from the operating system will then cause the graph to be printed.

```
plotter,ps
_pf='plot.dat'
_ti='Sine Wave'
x=indices(159)/10.
y=sin(x)
plot,x,y
closeplot
plotter,pc
end
```

The graphics commands are now to be found on disk in a file called plot.dat. Iconize IRT and output the file to the printer in normal fashion. A call to CLOSEGRAPH is necessary to close the file, thereby indicating that no overplot commands will be forthcoming.

Syntax

Command Files

A series of commands can be stored in a disk file and accessed from the keyboard. The commands are written in the file exactly as they would be entered from the keyboard. The command file is called by a name with a .irt extension (eg example.irt). The commands are executed when the user types the name of the file (without extension) and hits return. Typing *example* will cause PCL to find the file called example.irt and execute the commands therein. It will continue to execute the commands sequentially until it finds a line with the command *end*, at which point it will return to the user and await the next command. A command file can call another command file, and upon the execution of the second file, PCL returns to the first and continues execution of the first.

The command file is read into memory and compiled for use. If changes are made without leaving PCL, it must be recompiled with the COMPILE or RUN command. If a line of the command file is too long it can be broken into multiple lines. In a manner similar to UNIX, any line which terminates in a backslash will append the next line.

Directories

When PCL searches for a command or data file it first searches the current directory.   If it fails to find the file there, it then searches each of three user identified directories.   They are called _di, _d2, and _d3. They are set as a string assignment eg: _di= '\irt\test\'   This would make PCL search for a file called test.dat under the name \irt\test\test.dat.

Parameters

Information is stored by PCL in the form of parameters with names chosen by the user. Each parameter is one of five kinds. It can be integer, double precision, string, file pointer, or empty.

Empty parameters are those used or called without ever being defined. For example, if one types *print*,*param*, where *param* has never been defined, then the result will be ****.

Integer parameters are long integer variables and follow the rules of integer arithmetic.

Double precision parameters follow floating point arithmetic rules. Single precision is not supported by PCL, just as C emphasizes the use of double precision. String parameters contain one byte character variables, and are used to carry letters and numbers in ascii format. File pointers are used to identify input and output streams.

Each parameter has a dimension and length. Undefined variables have dimension and length of zero. Scalars have dimension zero and length of 1. Vectors have dimension of one and length equal to the number of elements. PCL also supports two dimensional arrays, which have dimension two and length equal to the total number of elements in the array.

Examples:

create integer scalar called *a* set value to 1	a=1
create double precision scalar called *b* set value to 1	b=1.
create integer vector *c* set elements to 1,2,3	c=[1,2,3]
create double precision vector *d* set elements to 0.,-24.,	d=[0.,-24.,1.e4]
create string vector *t* store the word String	t='String'
create two dimensional double array *x* fill 100 elements with zeroes	x=fltarr(10,10)

Statement Types

As shown by example in the previous section, an assignment statement stores a value or values in a parameter. If the parameter does not yet exist, it is created and its type set implicitly by the kind of value to be stored. If it already exists, then it is modified to store the new value. This includes a change of parameter type, dimension and length if appropriate.

Assignment Statement
An Assignment statement is identified by the use of an equal sign, as is standard in C and many other languages. PCL scans the command line for the equal sign. If found, then the command is treated as an assignment, otherwise it is assumed to be a procedure.

Procedure Statement
A procedure is similar to a subroutine. It is a set of commands defined either by PCL, or by the user that performs some task, but does not return a value for use in an assignment statement. For example, the command *print,a* is a procedure. It performs the task of printing the values stored in a parameter to the screen.

Functions
A function is a set of commands defined by PCL that returns a parameter for use or storage in an assignment statement. This can be a complicated set of instructions, or can be quite simple. *y = sin(x)* is an assignment statement. *sin(x)* is a function, the returned value of which will be stored in the parameter *y*. *print,y* is a procedure statement. It will print the values stored in *y* by the previous command.

Label Statement
A label statement is a position marker in the procedure file. It consists of a line with a single name, ending with a colon. Each label should be different from the others in the file.

GOTO Statement
A goto will transfer control to the position in the file identified by the label. Hence *goto,lab1* will cause pcl to look for a label statement *lab1:* and resume execution with the statement that follows *lab1:*

IF Statement
An IF statement provides a logical branch in the procedure. The word 'if' is followed by an expression in parentheses. If the value of the expression is greater than zero, the rest of the line after the parentheses is executed. If the value is zero or negative, then the line is not executed. A typical use would be in a line like: if(i == 7) goto,label1 which would cause control to be moved to the label statement called *label1.*

Comment Statement
Comments can be inserted in the procedure file. Any statement that begins with a semicolon will be ignored during the execution of pcl. e.g.:
;This is a comment will be ignored.

Pause and Continue Statements
Inserting the statement *pause* into the command file will cause execution to halt and give the interactive prompt irt> . The user can then give commands directly. When the user types *continue*, then execution will continue, and when the command file is completed, the usual irt> prompt is seen. Often in debugging a loop, there may be many pauses. Just typing the letter *c* will suffice instead of the whole word *continue*.

End Statement
The word *end* marks the end of the command file.   All text after this will be ignored.   When an *end* statement is encountered, control is returned to the level from which it was called.

Compile Statement
Compile, filename will cause PCL to find filename.pcl and compile it, replacing an old version if necessary.   It does not executes the new command file.   Compile is used when the file is changed without leaving the PCL environment.

Run Statement
Run, filename will cause PCL to find filename.pcl and compile it, replacing an old version if necessary.   It then executes the new command file. RUN is used when the file is changed without leaving the PCL environment.

Exit and Quit Statements
exit (or equivalently *quit*), wherever encountered will cause PCL to halt and a return to the operating system.

OPERATORS

Like most languages, PCL allows the user to perform mathematical operations on the parameters.

Operator	**Effect**
+	add
-	subtract
*	multiply
/	divide
[... , ...]	concatenate
==	logical equal
>	logical greater than
<	logical less than
&&	logical and
||	logical or
>=	logical greater than or equal to
<=	logical less than or equal to

Addition, subtraction, multiplication and division follow the usual rules of computer arithmetic. If the parameters involved are both integer, then integer arithmetic is used. If both are double precision, then double precision arithmetic is used. If one is integer and the other double precision, the contents of the integer parameter are converted to double precision before the operation.
The logical operators check to see if the statement is true, and return a 1 if true, or 0 if false. If both of the operands are integer, then the returned value is integer. If they are both double, or one is double and one integer, then the returned value is double.
Concatenate is used to append one vector or scalar to the end of another to form a new, longer vector. For example x = [2.,3.] will create a double vector of length 2 and store it in x. Similarly x = [x, 4.] will lengthen x to three elements. If one element is integer and one double, then the result will be double.
The only two operators that work on string variables are addition and concatenation. Both have the same effect, of appending one string to the end of another. For example:

```
s = 'First'
t = 'Second'
v = s + t
```

Will place the string 'FirstSecond' into v.

```
w = [v, ' third' ]
```

will put 'FirstSecond third' into w.

One of the chief advantages of PCL is its method of operating on vectors and arrays. Each element of the vector is operated upon without having to create a do loop to explicitly work through the array. For example, *x = [1, 2, 3]* followed by print, *2*x* will cause 2 4 6 to be printed on the screen.
When a scalar operates on a vector, then each element operates with the scalar. When two vectors are operating, then the operation goes on an element by element basis. For example [1, 2, 3] + [4, 5, 6] will yield [5, 7, 9]. If the vectors are not of the same length, then the longer will be truncated to the length of the shorter.
When a vector is used as the calling parameter of a function, then the function returns a vector of the same length as the calling parameter. Each element is operated upon by the function individually. This is a very powerful syntax for user manipulation of information.

Functions and Procedures

TOTAL
WHERE

ABS(x)

x = parameter to be converted to integer type

ABS converts x to the absolute value of x and returns it as an integer parameter. If x is string or undefined it returns undefined.   It is the same as FABS().

ACOS(x)

x = vector input

ACOS returns the arc-cosine of a parameter in radians, operating on each element individually.   If *x* is integer, it is converted to double, and the function returns a double variable.

ASIN(x)

x = vector input

ASIN returns the arcsine of a parameter in radians, operating on each element individually.
If *x* is integer, it is converted to double, and the function returns a double variable.

ATAN(x)

x = vector input

ATAN returns the arctangent of a parameter in radians, operating on each element individually. If *x* is integer, it is converted to double, and the function returns a double variable.

ATAN2(y, x)

x = vector input
y = vector input

ATAN2 returns the arctangent of an ordered pair in radians, operating on each pair of elements individually. If *x* or *y* is an integer, it is converted to double, and the function returns a double variable.

AVERAGE(x)

x = vector input

AVERAGE returns the average of the elements of a parameter. If *x* is integer type it is converted to double before averaging.

CLOSEPLOT
CLOSEPLOT closes the disk file to which plotting commands were being sent. Before closing it inserts any needed printer commands into the file.

CONTINUE
CONTINUE causes continuation of execution of a command file interrupted by PAUSE. The single letter C can be used as an abbreviation for CONTINUE.

COS(x)

x = vector input

COS returns the cosine of a parameter, operating on each element individually. If *x* is integer, it is converted to double, and the function returns a double variable. COS assumes the input is in radians.

DATE()

DATE is a function that returns a character string containing the date from the operating system.

END
END is used as the last command of a PCL command file. It signals PCL that the last command has been reached. All text after END in the command file is ignored by PCL. During command execution, when END is encountered, then control is returned to user. In the case of a procedure, END signals return to the calling routine.

ERASE
ERASE causes the screen to be erased.

EXIT
The EXIT command is the same as the QUIT command. It is entered by the user when the PCL session is complete. It causes the program to terminate and control return to the operating system.

EXP(x)

x = vector input

EXP exponentiates a parameter, operating on each element individually. If *x* is integer, it is converted to double, and the function returns a double variable.

FABS(x)

x = parameter to be converted to integer type

FABS converts x to the absolute value of x and returns it as an integer parameter. If x is string or undefined it returns undefined.   It is the same as ABS().

FCLOSE, lun

lun = logical unit number of file

This procedure closes the file identified by *lun*.

FIX(x)

x = parameter to be converted to integer type

This function converts to x to integer type and returns it as an integer parameter. If x is string or undefined it returns undefined.   If x is integer or double it returns integer. Conversion follows the rules of ANSI C, so a double is truncated to form a integer (not rounded).

FLOAT(x)

x = parameter to be converted to double type

This function converts x to double type and returns it as a double parameter. If x is string or undefined it returns undefined.   If x is integer or double it returns double.

FLTARR(n)

n = length of array

FLTARR is a function that returns a double precision vector of length n.   The elements of the vector are all set to 0.

FPRINTF, lun, x1, x2, ...

lun = logical unit number of file

xn = parameters to be printed

This procedure prints the contents of the parameter to the file identified by the integer parameter *lun*. As many parameters as desired can be listed, and they will be printed in order. Attempting to print an undefined parameter will yield asterisks. *lun* is usually set by a call to the function FOPEN.

FOPEN(name, rw)

name = name of file to be opened

rw = read or write file

This function returns a pointer parameter which is a file identifier. *name* is a char string which identifies the file. *rw* is a character string which identifies whether the file is for reading or writing. 'r' will yield a file for reading; 'w' will give writing. Note, a pointer parameter can be stored, but not operated upon.

```
lun=fopen('data.dat','r')
i=readi(lun)
print,i
fclose,lun
end
```

HISTOGRAM(x, bin, xmin, xmax)

x = vector of values to be histogrammed
bin = bin size (1.)
xmin = lower end of range to be histogrammed (0.)
xmax = upper end of range to be histogrammed (max value of x)

HISTOGRAM is a function that returns an integer vector. The values are the number of elements of x that fall in each bin. *bin* is used to control the size of the bins, while *xmin* and *xmax* control the range of operation. Values of x that do not fall between *xmin* and *xmax* are ignored.

```
;EXAMPLE
a=sqrt(indices(1000))
h=histogram(a,2.5,5.,40.)
nbins=fix((40.-5.)/2.5)+1
b=indices(b)*2.5+5.
plot,b,h
end
```

INDICES(n)

n = length of array

INDICES is a function that returns a double precision vector of length n.   The elements of the vector are all set to the number of the element, i.e. the first element is 0., the second is 1., third 2., etc.   This is very useful for generating functions.

INTARR(n)

n = length of array

INTARR is a function that returns an integer vector of length n.   The elements of the vector are all set to 0.

LOG(x)

x = vector input

LOG returns the natural logarithm of a parameter, operating on each element individually.
If *x* is integer, it is converted to double, and the function returns a double variable.

LOG10(x)

x = vector input

LOG10 returns the base 10 logarithm of a parameter, operating on each element individually. If *x* is integer, it is converted to double, and the function returns a double variable.

MAX(x)

x = vector to be searched

MAX searches the vector x and returns a scalar parameter of value equal to the largest element of x. It functions with integer and double types, and returns a scalar of the same type as the vector.

MIN(x)

x = vector to be searched

MIN searches the vector x and returns a scalar parameter of value equal to the smallest element of x. It functions with integer and double types, and returns a scalar of the same type as the vector.

LENGTH(x)

x = parameter

LENGTH operates on a vector of any type. It returns an integer scalar whose value is the length of the vector. A scalar has length 1, as does a vector of length 1. An undefined vector causes a value of 0 to be returned.

```
;EXAMPLE
 x=[1.,2.,3.]
print,length(x)
3
; y is undefined
print,length(y)
0
end
```

OPLOT, x, y, color

x = x values of ordered pairs

y = y values of ordered pairs

color = color of each element

OPLOT is the procedure used to plot a second series of points to the screen or hard copy device.   It is explained in detail in the chapter on plotting.

PAUSE
PAUSE halts the execution of a command file and gives control to the user with the PCL1> prompt. User commands can then be given. When the user types CONTINUE (or simply C), then execution of the command file continues where it left off. If the user, during a pause, runs another routine containing a pause command, then the prompt PCL2>, for the second level of pause is encountered. CONTINUE will cause execution of the second level routine to continue. When it is completed control returns to the PCL1> level.

PLOT, x, y, color

x = x values of ordered pairs
y = y values of ordered pairs
color = color of each element

PLOT is the procedure normally used to plot data to the screen or hard copy device.   It is explained in detail in the chapter on plotting.

POW(x, y)

x = vector input

y = vector input

POW returns x raised to the power of y, operating on each element individually. If *x* or *y* is integer, it is converted to double, and the function returns a double variable.

PRINT, x1, x2, ...

xn = parameters to be printed

This procedure prints the contents of the parameter to the user's screen. As many parameters as desired can be listed, and they will be printed in order. Attempting to print an undefined parameter will yield asterisks.

PRINTF, x1, x2, ...

xn = parameters to be printed

This procedure prints the contents of the parameter to the user's screen. As many parameters as desired can be listed, and they will be printed in order. Attempting to print an undefined parameter will yield asterisks. It is identical to PRINT. PRINTF is available to maintain consistency of notation with C.

QUIT
The QUIT command is the same as the EXIT command. It is entered by the user when the PCL session is complete. It causes the program to terminate and control return to the operating system.

READD(filenum, nvalues)

filenum = file pointer parameter

nvalues = number of numbers to read in (1)

This function allows the user to read values from an ascii disk file into a double precision parameter. *filenum* is the pointer to the file created by the call to FOPEN. *nvalues* tells PCL to read more than one value into the array. If READD is called with only the file pointer, then only one value will be read.

READI(filenum, nvalues)

filenum = file pointer parameter

nvalues = number of numbers to read in (1)

This function allows the user to read values from an ascii disk file into an integer parameter. *filenum* is the pointer to the file created by the call to FOPEN. *nvalues* tells PCL to read more than one value into the array. If READI is called with only the file pointer, then only one value will be read.

READS(filenum)

filenum = file pointer parameter

This function allows the user to read ascii character strings from a disk file into an character parameter. *filenum* is the pointer to the file created by the call to FOPEN. The parameter will take one line at a time; all the characters until an end of line or end of file is encountered.

SIGMA(x)

x = vector input

SIGMA returns the standard deviation of the elements of a parameter. If *x* is integer type it is converted to double before averaging.

SIN(x)

x = vector input

SIN returns the sine of a parameter, operating on each element individually. If *x* is integer, it is converted to double, and the function returns a double variable. SIN assumes the input is in radians.

SQRT(x)

x = vector input

SQRT returns the square root of a parameter, operating on each element individually. If *x* is integer, it is converted to double, and the function returns a double variable.

STRING(x)

x = vector for conversion to character string

STRING converts integer and double vectors and scalars to character strings. This is useful for printing the values of parameters in user defined formats.   If x is string type, it returns the value unchanged.   If the vector has a length greater than one, it converts all the elements and returns them with spaces in between.

TAN(x)

x = vector input

TAN returns the tangent of a parameter, operating on each element individually. If *x* is integer, it is converted to double, and the function returns a double variable. TAN assumes the input is in radians.

TOTAL(x)

x = vector input

TOTAL returns the sum of the elements of a parameter, operating on each element individually.

WHERE(x)

x = data array

WHERE is a function that returns an integer vector. The elements of the vector are a list of the indices of x where x is greater than 0. x is often the output of a logic statement. For example, WHERE(indices(4) > 1) will return the vector [2,3].