

SAND##-####  
Unlimited Release  
**DRAFT**  
Version January 1, 1996

Distribution  
Category \*\*-##

# **PFIDL Users Guide**

## **Procedures for the Analysis and Visualization of Data Arrays**

L. Paul Mix  
Electromagnetics and Plasma Physics Analysis  
Sandia National Laboratories  
Albuquerque, NM 87185 -1186

### **Abstract**

PFIDL is an analysis and visualization package written to assist scientists in obtaining a better understanding of experimental and theoretical data and for the graphical generation of theoretical code input. PFIDL provides extensions and an improved user interface for the IDL<sup>®</sup> software package from Research Systems, Inc. with routines written in Fortran, C, and the IDL<sup>®</sup> procedural language. In addition to the standard ASCII dataset formats, PFIDL provides a convenient interface to PFF, Exodus-II, PDS, PDB, and ACIS data formats, and has convenient output routines for the GIF, TIFF, MPEG and PostScript formats. Full IDL functionality is maintained to facilitate redundant analysis of multiple datasets. Special purpose analysis routines, the rich selection of input routines, and extensive comparison, mathematical and display routines, facilitate the rapid understanding of experimental and theoretical datasets. PFIDL also contains a Fortran library and graphical user interface for IDL that facilitates the use of the PFIDL graphics and analysis procedures in Fortran programs while maintaining the command recall and command editing functionality of IDL. On-line documentation of all routines, with examples, allows new users quick access to the powerful PFIDL routines using either normal IDL or the Fortran callable version.

# Acknowledgment

Implementation of the Portable File Format would not have been possible without the assistance and support of David B. Seidel. The command syntax is based on the IDR analysis code developed by W. B. Boyer in the 1970's.

PFIDL Version 1.5

Copyright 1999 Sandia Corporation. Under the terms of Contract DE-AC04-94AL85000, there is a non-exclusive license for use of this work by or on behalf of the U.S. Government.

NOTICE: The United States Government is granted for itself and others acting on its behalf a paid-up, non-exclusive, irrevocable worldwide license in this data to reproduce, prepare derivative works, and perform publicly and display publicly. Beginning five (5) days from November 26, 1999, the United States Government is granted for itself and others acting on its behalf a paid-up, non-exclusive, irrevocable worldwide license in this data to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

NEITHER THE UNITED STATES GOVERNMENT, NOR THE UNITED STATES DEPARTMENT OF ENERGY, NOR SANDIA CORPORATION, NOR ANY OF THEIR EMPLOYEES, MAKES ANY WARRANTY, EXPRESSED OR IMPLIED, OR ASSUMES ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETENESS, OR USEFULNESS OF ANY INFORMATION, APPARATUS, PRODUCT, OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE WOULD NOT INFRINGE PRIVATELY OWNED RIGHTS.

# PFIDL Users Guide

## Procedures for the Analysis and Visualization of Data Arrays

### Table of Contents

The PFIDL Interface to IDL .....	1-1
Introduction.....	1-1
IDL/ PFIDL Basics .....	1-1
Variables .....	1-1
Expressions .....	1-2
Procedures and Functions .....	1-3
Math Routines.....	1-4
Filters .....	1-4
Convolutions.....	1-5
PFF Files .....	1-5
Getting Started .....	1-6
PFIDL Analysis .....	1-7
Sample First Session.....	1-7
Functional Command Summary .....	2-1
Filter Routines.....	2-1
Help.....	2-1
Math Routines.....	2-1
Miscellaneous .....	2-2
PFF Utilities .....	2-2
QuickSilver / Image Analysis.....	2-3
WDF Input/Output.....	2-3
WDF Utilities.....	2-3
Examples of PFIDL .....	3-1
Read waveform arrays and plot the waveforms.....	3-1
Calculate power from current and voltage.....	3-1
Generate magnetic stream function .....	3-1
View a scanned CR-39 Image .....	3-2
Plot lineouts from an image file.....	3-2
Waveform analysis .....	3-3
Command Procedure for QUICKSILVER Post-Processing.....	3-3
Command Procedure for QUICKSILVER Post-Processing.....	3-5
Command Reference Dictionary .....	4-1
ACTIVATE .....	4-1
ADD.....	4-2

ADD_FRAMES .....	4-3
ADDFR .....	4-3
ADDSTR .....	4-5
ADD_COYOTE.....	4-7
ARR2STR.....	4-7
AVE .....	4-8
BAP.....	4-9
BAR .....	4-9
BLINE.....	4-10
Subtract a constant value from a waveform. Baseline can be set by clicking left mouse button or by entering a value. Once a baseline is set the baseline will follow the cursor. Baseline is accepted with the middle mouse button. Program exits. Selected baseline is reset with the right mouse button. Exit/Quit button will exit with no baseline. ....	4-10
BORDER .....	4-10
BOX .....	4-11
CHA .....	4-12
CHASTR.....	4-13
CLOSEPFF .....	4-16
COM .....	4-16
COMP_GEO.....	4-17
COMTE .....	4-18
COND_RANGE .....	4-20
CONV .....	4-21
CREATEPFF .....	4-21
CUR .....	4-22
CURP .....	4-22
CYL2CART.....	4-23
DECON.....	4-25
DELSTR .....	4-25
DELWDF.....	4-25
DIF .....	4-26
DIGITIZE .....	4-26
DIR.....	4-27
DIRP .....	4-28
DIRPFF .....	4-28
DIRS .....	4-30
DIRW .....	4-30
DIV .....	4-31
DIVSTR .....	4-32
Droop .....	4-34
ENDPFF.....	4-35
ENDPOST .....	4-35
ENVELOPE.....	4-35
FACTORS .....	4-37
FALLT .....	4-38
FILM_EXP .....	4-38

FFTF .....	4-39
FFTW .....	4-40
FILT .....	4-41
FULFNAM .....	4-43
FWHM .....	4-43
GET_ARRAY .....	4-44
GET_DIR .....	4-44
GET_DSET .....	4-45
GET_FCUP .....	4-47
GET_MATCH .....	4-48
GET_MAXSTR .....	4-50
GET_MAXWDF .....	4-50
GET_RESP .....	4-51
GET_STR_VALUE .....	4-51
GET_TIME .....	4-60
GET_VALUE .....	4-60
GET_VPROP .....	4-61
GET_WDF .....	4-62
GETF .....	4-63
GETX .....	4-63
GETY .....	4-63
GETYF .....	4-64
GTITLE .....	4-64
HAK .....	4-65
HEADER .....	4-65
HIP .....	4-67
IDL Functions .....	4-67
I2S .....	4-68
I2W .....	4-68
IMAGE .....	4-69
INT .....	4-71
INTSTR .....	4-71
INVERTCT .....	4-74
IPROP .....	4-74
JOINW .....	4-75
JOINSTR .....	4-76
KILL_PART .....	4-76
LAB .....	4-77
LABEL .....	4-77
LEGEND .....	4-78
LHELP .....	4-80
LIM .....	4-80
LINSTR .....	4-81
LOADXYCT .....	4-83
LOP .....	4-83
LPRINT .....	4-84

LS.....	4-85
MAXSTR.....	4-85
MINSTR.....	4-86
MED.....	4-87
MID.....	4-87
MIRROR.....	4-88
MKSYMBOL.....	4-89
MN.....	4-89
MUL.....	4-90
MULSTR.....	4-91
MX.....	4-92
MYTITLE.....	4-93
NEWEND.....	4-94
OPENPFF.....	4-94
PAD.....	4-95
PARSER.....	4-96
PE2PFF.....	4-96
PFF_DIAG.....	4-98
PFF_INFO.....	4-99
PFF_REGRID.....	4-100
PIXPLOT.....	4-101
PLO.....	4-104
PLOTCON.....	4-109
PLOT_CUBE.....	4-110
PLOTFLD.....	4-113
PLOTGRD.....	4-121
PLOTPAR.....	4-122
PLOTSTR.....	4-125
PLOTVEC.....	4-125
PLOTWT.....	4-128
PRESET.....	4-129
PRIAR.....	4-129
PS2X.....	4-130
PUT_ARRAY.....	4-130
PUTX.....	4-130
PUTY.....	4-131
PWD.....	4-131
RE.....	4-131
READ_PE.....	4-133
READASC.....	4-135
READCON.....	4-136
READFLD.....	4-137
READGRD.....	4-139
READHDR.....	4-140
READHIST.....	4-141
READIFL.....	4-142

READIMAGE .....	4-143
READNF3 .....	4-143
READNGD .....	4-145
READNI3 .....	4-147
READNV3 .....	4-148
READPAR .....	4-150
READPFF .....	4-151
READPOP .....	4-154
READRGA .....	4-155
READSIG .....	4-155
READSTR .....	4-156
READTEK2 .....	4-157
READTEK3 .....	4-158
READTK .....	4-159
READTK2 .....	4-161
READUF1 .....	4-162
READUF3 .....	4-164
READVTX .....	4-165
READWD3 .....	4-166
REASC .....	4-167
RESTORECT .....	4-168
RESTOREPFIDL .....	4-168
REVIDA .....	4-168
RISSET .....	4-169
RMBASE .....	4-169
ROTSTR .....	4-170
RTP .....	4-172
RTPSTR .....	4-173
S2I .....	4-174
SAVECT .....	4-174
SAVEPFIDL .....	4-174
SCALE_REGRID .....	4-175
SCANF .....	4-177
SCANGRD .....	4-178
SCANP .....	4-179
SCANS .....	4-179
SCANV .....	4-180
SET_DRAWSIZE .....	4-181
SET_MAXSTRARRAY .....	4-182
SET_MAXWDFARRAY .....	4-182
SETDEFAULT .....	4-182
SETDIRPFF .....	4-183
SETPFF .....	4-183
SGIF .....	4-183
SHELP .....	4-184
SHOWPFF .....	4-184

SLICE .....	4-185
SMO .....	4-186
SP2XYZ .....	4-187
SPEC .....	4-187
SPICT .....	4-188
STARTEPS .....	4-188
STARTGIF .....	4-189
STARTP .....	4-190
STARTPCL .....	4-191
STARTPOST .....	4-192
STIFF .....	4-192
STOPPING .....	4-193
STRCOMMENT .....	4-194
STRVALID .....	4-195
SUB .....	4-196
SUBSTR .....	4-197
SUM .....	4-198
SWAP .....	4-200
SWAPCT .....	4-200
SYMBOL_LIST .....	4-200
THERMOMETER .....	4-201
TLPARAM .....	4-204
TOTFR .....	4-205
TRIFL .....	4-207
TRIFLW .....	4-209
TSH .....	4-212
USERQSINIT .....	4-213
VIDA2STR .....	4-213
VIDA_AXIS .....	4-214
W132 .....	4-214
W2I .....	4-214
W80 .....	4-215
WDCOMMENT .....	4-216
WDEDIT .....	4-216
WDERR .....	4-216
WDFIT .....	4-218
WDFIT3 .....	4-218
WDSLPIT .....	4-219
WDVALID .....	4-220
WHATS_NEW .....	4-221
WHELP .....	4-221
WIN .....	4-222
WRI .....	4-222
WRIASC .....	4-223
WRIIFL .....	4-224
WRIMSD .....	4-224

WRISTR .....	4-225
WRIVTX .....	4-226
WRITEIFL .....	4-226
WRITEPPF .....	4-227
WRITETK .....	4-228
X2PS .....	4-229
XCUR .....	4-229
XCURW .....	4-230
XFR .....	4-231
XFRSTR .....	4-231
XLINEOUT .....	4-232
XPLO .....	4-232
XRESET .....	4-233
YLINEOUT .....	4-233
ZOOMW .....	4-234
<b>The PFIDL Structures .....</b>	<b>A-1</b>
PFIDL Structures .....	A-1
<b>Command Syntax .....</b>	<b>B-1</b>
Comparison of WDF and structure commands.....	B-1
<b>Quicksilver Application Datatypes .....</b>	<b>C-1</b>
Quicksilver Application Datatypes.....	C-1
<b>TQ Build .....</b>	<b>D-1</b>
Overview.....	D-1
Conductor Definition .....	D-2
Graphical User Interface.....	D-2
ASCII Data File .....	D-2
DXF File .....	D-3
IDL Functions.....	D-3
Grid Generation .....	D-4
Mathematical Overview.....	D-4
Examples.....	D-5
Grid Details.....	D-6
TQBuild Command Reference Dictionary .....	D-9
READDXF.....	D-9
QSEdit .....	D-10
QSFIT .....	D-11
QSGRID.....	D-11
QSWRITE.....	D-15
TQCOND.....	D-15
TQEDIT .....	D-16
TQFILL.....	D-16
TQFILTER.....	D-17
TQFIT .....	D-17
TQGRID .....	D-18
TQHELP.....	D-19

TQRESTORE .....	D-20
TQSAVE.....	D-20
Convolutions .....	E-1
References .....	R-1

## 1.0 The PFIDL Interface to IDL

### 1.1 Introduction

Much of our understanding of experimental data and computer simulations comes from the analysis waveform datasets and the display of 2 and 3 dimensional field or particle datasets. Waveform, or time-history, datasets are linear arrays of data in which a single physical parameter, such as voltage, current, particle density, temperature,... is expressed as a function of a single variable, such as time or distance. To simplify the analysis of both waveform and multidimensional data, the PFIDL (PFF - IDL) interface has been written in IDL<sup>1</sup> to allow the easy mathematical manipulation and visualization of these datasets. Full IDL functionality is maintained to permit the simultaneous analysis of image or volumetric type data and to allow the FOR, IF, WHILE, and CASE constructions to be used for performing redundant operations on multiple datasets. The Portable File Format (PFF)<sup>2</sup>, developed to make the transport of large, mixed integer and floating point data files between various computers, is fully supported by the PFIDL utilities. The PFIDL procedures are typically command driven to facilitate automation of the analysis process; however commands which by their nature are interactive, such as the on-line help facility, make use of the IDL widgets.

This user's manual describes the PFIDL procedures and provides enough information about IDL to allow the user to immediately begin analysis of his data. Chapter 1 provides an introduction to IDL and PFIDL. Chapter 2 provides a brief overview of the available routines in PFIDL separated according to their function. Chapter 3 provides some examples of typical analysis sessions. Chapter 4 is a command reference dictionary with a complete alphabetical listing of each procedure and detailed descriptions of the parameters and keywords and some examples of the usage. Revisions of this manual will lag the software development. On-line help is accessed with the LHELP command and contains the most current information.

### 1.2 IDL/ PFIDL Basics

A brief overview of the IDL/ PFIDL package is presented in this section. Some previous experience with some other high level language is assumed. For a more detailed discussion of the IDL operation and syntax, the reader is referred to the IDL Users Guide. The variables used by IDL/ PFIDL are first presented. IDL expressions, functions, and procedures are then presented.

#### 1.2.1 Variables

IDL supports byte, short and long integer, single and double precision floating point, complex, and string data types. These data types can be scalars, arrays, or aggregate structures. Variable assignment is with a simple arithmetic statement of the form: *variable* = *value*, where *variable* is the IDL variable name and *value* is the corresponding value. Memory is dynamically allocated; *variable* will automatically take the form of *value* both regarding its type (real, integer, complex,...) and its structure (scalar, array, or structure element) and attributes previously assigned to *variable* are

destroyed. IDL variable names must begin with a letter and can have from 1 to 15 characters including the underline and dollar sign. IDL variable names are case insensitive. IDL arrays may be subscripted to select one or more elements. The elements are numbered starting with 0 as the first element and the first element of multi-dimensional arrays, like FORTRAN, but unlike linear algebra and C, varies the fastest. Information about current IDL variables can be obtained by entering: **HELP** to examine all variables or by entering: **HELP, variable** to look at a specific variable. Array values can be determined using: **PRINT, variable (0: 11)** to examine the first 12 values; using no subscript prints the entire array.

PFIDL variables are used to simplify the analysis of datasets and can be considered to represent structures, or collections, of several different types of IDL variables. In the present release of PFIDL there are two types of PFIDL variables and each type is represented by a number from 1 to 82. The first type of PFIDL variable represents a dataset of a single independent variable (a waveform array) the second type represents a dataset of two or more independent variables. An example of the first type of PFIDL variable is voltage as a function of time. An example of the second type is electric field as a function of x and y. An extension of the second type of variable is to allow multiple dependent variables to be stored in a single PFIDL multidimensional variable. The default form of a waveform dataset is to be characterized by a number of points, np, an initial abscissa, an abscissa increment, and np ordinate values. This form will be referred to as a Waveform Data Format (WDF) array. In the present implementation, waveform datasets can also be described as order pairs of data for the display of complex shapes. In addition to the actual data, strings describing the data, axis labels, and other data labels are stored with the data. These strings are read directly from PFF datasets and can be modified to improve the appearance of graphs. PFIDL variables can only be modified using the PFIDL functions and procedures. For example to add variable, or array, 1 to variable, or array, 2 and store the result in array 3 for all times which the arrays 1 and 2 have an overlap, use the command: **add, 1, 2, 3**. Corresponding to the **HELP** command for IDL is the **WHELP** command for PFIDL waveform variables and **SHELP** for the multidimensional PFIDL variables. **WHELP** and **SHELP** will list the PFIDL variable numbers and the corresponding dataset comments for all PFIDL arrays containing data. To look at the actual values stored in the PFIDL variable number with the command; for example, **SHELP, 3** or **WHELP, 3** to look at the actual range of data in these arrays.

IDL variables can be used to represent PFIDL variables. If a variable is passed to a PFIDL procedure to represent an output waveform dataset and is undefined, then PFIDL will set the variable equal to the lowest unused WDF array number. For example the expression: **add, 1, 2, 3** could be changed to **add, 1,2, sum**; if sum is undefined, then it will be set to the value of the lowest unused WDF array number; if sum is defined but not equal to a valid WDF array, then it will be set to 1 on exiting the **add** command.

### 1.2.2 Expressions

IDL variables and constants may be combined, using operators and functions, into expressions. Unlike FORTRAN, IDL expressions may be scalar or array-valued. The number of operators in IDL are quite numerous; they include not only the standard addition (+), subtraction (-), multiplication (\*), division (/), exponentiation (^), relations (EQ, NE, GT, GE,...) and Boolean arithmetic (AND, NOT, OR, XOR), but also operators to

find maxima (>) and minima (<), to select scalars and subsets of arrays from arrays (subscripting), to concatenate scalars and arrays to form new arrays ([ ]), and to perform matrix multiplication (#). Functions, which are operators in themselves, exist in IDL for performing data smoothing, shifting, transforming, evaluation of transcendental functions,.... The general form of an expression is: *variable* = *value*, where *variable* is the IDL variable name and *value* is a constant, variable, expression operator, function or a combination of these types. The structure of an expression, and thus the structure of *variable*, may be either a scalar or an array, and, unlike many other languages, the type and structure cannot be determined until the expression is evaluated. Care must be exercised when writing programs. For example, a variable may be a scalar byte at one point in a program while at a later point it may be set to a complex array.

An important feature of IDL, not available in FORTRAN, which is frequently useful in working with waveform data arrays is the concatenation operator. Operands enclosed in square brackets and separated by commas are concatenated to form larger arrays. For example, the expression, [*a*, *b*], is an array formed concatenating the first dimensions of *a* and *b*, which may be either scalars or arrays. (Note: String concatenation is accomplished with a + operator; that is, 'ABC' + 'DEF' = 'ABCDEF').

The details of expressions are covered in the IDL manual and will not be covered in any more detail here. The PFIDL arrays cannot be manipulated with standard IDL expressions, but rely on functions and procedures, described in the next section, for defining and modifying the values.

### 1.2.3 Procedures and Functions

IDL procedures and functions are modules that perform well defined tasks, frequently on a set of parameters. Two types of parameters can be used by procedures and functions and are designated by their position or by a keyword in the calling statement. Parameters may be an expression or a variable name. Functions are used just as you would use a variable; for example:

```
a = function_name(a, b, c, max = d, e, /plot),
```

where *a*, *b*, *c*, and *e* are positional parameters and *d* is a keyword parameter for the function, *function\_name*. A similar call for a procedure would be:

```
procedure_name, a, b, c, max = d, e, /plot.
```

The keyword parameters are preceded by a keyword and an equal sign; the keyword can be shortened to its shortest unambiguous abbreviation. Keyword parameters can also be specified with the syntax, */keyword*, which is equivalent to *keyword* = 1. The positional parameters, or plain arguments, have no keywords; the *n*<sup>th</sup> positional parameter is matched with the *n*<sup>th</sup> actual positional parameter.

A procedure or function may be called with less arguments than were defined in the procedure or function; parameters not used are set to be *undefined*. On exiting a procedure or function, the formal parameters are copied back to the actual parameters, provided that they were not expressions or constants. Thus, parameters may be inputs to a program, they may be outputs in which the values are set or changed, or they may be both inputs and outputs.

If a PFIDL procedure or function dies for any reason, use the command, **RECALL**, before trying the command again.

## 1.3 Math Routines

PFIDL procedures and functions rely on the internal IDL mathematical routines for most applications. The techniques employed for the frequency domain filter routines and the convolution routines are described in this section.

### 1.3.1 Filters

Six filter functions are provided for WDF arrays: low pass, high pass, band pass, band reject Fourier filters, a median filter, and a boxcar filter. The first 4 filter types, obtained using the `FILT` command, utilize the Fast Fourier Transform (FFT) function of IDL to obtain the frequency components of the waveform which is then multiplied by an attenuation array. The Inverse Fourier Transform is then taken and the resulting array returned to the user. The FFT routine assumes a periodic function with a period equal to the domain of the array. If the first and last points differ, then distortions will be introduced in most cases. A boxcar type smooth routine is obtained with the `SMO` command and median filtering with the `MED` command. See `FILT`, `MED` or `SMO` for more details.

If FFT represents a Fourier filter and IFT, its inverse, then the filtered array  $b$  is defined by the following:

$$b = \text{IFT} ( \text{filter} * \text{FFT}(a) )$$

where *filter* is defined below or supplied by the user.

Assume  $np$  points in the array with spacing  $dt$ , where  $\wedge$  indicates an exponent and  $dist$  is an array of mode numbers with  $np$  elements defined as:  $dist = [0, 1, 2, 3, \dots, np/2, \dots, 3, 2, 1]$ . *mode*, *width* and *order* are specified by the user and used to calculate the following filter functions.

The low pass filter is defined by the following:

$$\text{Butterworth: } \text{filter} = 1 / (1 + ( \text{dist} / \text{mode} ) ^ (2 * \text{order}))$$

$$\text{Exponential: } \text{filter} = \text{EXP} ( - ( ( \text{dist} / \text{mode} ) ^ \text{order} ) )$$

$$\text{Ideal: } \text{filter} = 1 \text{ for } \text{dist} \leq \text{mode}, 0 \text{ for } \text{dist} > \text{mode}$$

Similarly the high pass filter is defined by the following:

$$\text{Butterworth: } \text{filter} = 1 / (1 + ( \text{mode} / \text{dist} ) ^ (2 * \text{order}))$$

$$\text{Exponential: } \text{filter} = 1 - \text{EXP} ( - ( ( \text{dist} / \text{mode} ) ^ \text{order} ) )$$

$$\text{Ideal: } \text{filter} = 0 \text{ for } \text{DIST} \leq \text{MODE}, 1 \text{ for } \text{dist} > \text{MODE}$$

Similarly the band type filter is defined by the following where:

$$\text{Butterworth: } \text{filter} = 1 / (1 + ( \text{dist} * \text{width} / ( \text{dist} ^ 2 - \text{mode} ^ 2 ) ) ^ (2 * \text{order}))$$

$$\text{(for band pass: } \text{filter} = 1.0 - \text{filter})$$

$$\text{Exponential: } \text{filter} = \text{EXP} ( - ( ( ( \text{dist} ^ 2 - \text{mode} ^ 2 ) / ( \text{dist} * \text{width} ) ) ^ (2 * \text{order}) ) )$$

$$\text{(for band reject: } \text{filter} = 1.0 - \text{filter})$$

$$\text{Ideal: } \text{filter} = 0 \text{ for } \text{dist} < \text{mode} - \text{width} / 2 \text{ or } \text{dist} > \text{mode} + \text{width} / 2$$

$$= 1, \text{ OTHERWISE}$$

$$\text{(for band reject: } \text{filter} = 1.0 - \text{filter})$$

**Note:** For exponential and Butterworth band type frequency, the true center mode,  $cm$ , is given by  $cm = \text{mode} * \text{SQRT} (1 + ( \text{width} / (2 * \text{mode}) ) ^ 2)$ .

### 1.3.2 Convolutions

Most experimental waveform data is that is recorded and stored,  $c(t)$ , is actually a convolution of the actual data,  $d(t)$ , with a response function,  $r(t)$ . The PFIDL convolution routines, CONV, DECON, and GET\_RESP, are based on the following relation between the Fourier transform of these three convolution quantities:

$$C(f) = D(f) \times R(f), \quad (\text{EQ 1})$$

where  $C(f)$  is the complex Fourier transform of  $c(t)$ ,  $D(f)$  is the complex Fourier transform of  $d(t)$ , and  $R(f)$  is the complex Fourier transform of  $r(t)$ . In the solution for  $d$  and  $r$ , an optional Wiener-filtered convolution is available of the following form for the deconvolution and for calculating the response function:

$$D(f) = C(f) / R(f) / (1 + 0.005 F \times (M / |R|)^2) * (1 + 0.005 * F), \quad (\text{EQ 2})$$

$$R(f) = C(f) / D(f) / (1 + 0.005 F \times (M / |R|)^2) * (1 + 0.005 * F), \quad (\text{EQ 3})$$

where  $F$  is typically 1,  $|R|$  is the magnitude of  $R(f)$ , and  $M$  is the maximum value of  $|R|$ . This form was suggested in a LLNL report<sup>3</sup>. Users are warned that even with this filter, large amplitude noise will be generated if  $c(t)$  contains high frequency noise such as from the digitizer or other sources close to the recording point. In addition large values of  $F$  will degrade, or attenuate, the signal. See appendix E for an example of these procedures.

## 1.4 PFF Files

The PFF file format was developed in 1989 by David Seidel to provide a data format which was not only compact but also binary compatible between the various computer systems in use at Sandia. Each file has its own directory structure which is appended to the end of the file. The dataset structure includes a number of character string labels which have an unlimited (actually, a limit of 65535) length for not only providing the name of the signal but also each of the axes associated with the data and for a comment and a data type label. The IDL interfaces to PFF support the following dataset types which have been assigned the PFF dataset type numbers on the left.

1. scalar data on a uniform 3-D grid (UF3),
2. scalar data on a uniform 1-D grid (similar to WDF data) (UF1),
3. scalar data on a non-uniform 3-D grid (NF3),
4. vector data on a non-uniform grid (NV3),
5.  $m \times n$  vertex data -  $m$ -dimensional vertices with  $n$ -attributes per vertex (VTX),
6. integer list and a two float lists (IFL),
7.  $n$ -dimensional vectors on a region of  $m$ -dimensional space (NGD),
8. nonuniform 3D grid (NG3),
9. integer data on a nonuniform 3D grid (NI3).

These dataset types are written with a standard precision of 16 bits; the data files are about a factor of two smaller than standard floating point binary files but have adequate precision for most applications. The maximum error in encoding the data is

$\pm 7.6 \times 10^{-6}$  of the dataset range. This precision is accomplished by encoding with each set of data (and each component of the vector for PFF type 4) an offset and a scale factor. These floating point numbers are encoded with 30 bits of precision for the fraction and 15 bits of precision for the exponent; thus the full Cray range can be encoded with PFF data types.

Except for the vertex data type, each of the dataset types supports a multiple block structure within each dataset. For example, suppose a dataset contained dose information from a photon transport code. One block of data might be for points near the source and additional blocks for regions further from the source. By breaking the problem into blocks, the dynamic range can be extended beyond the 16 bits (to  $10^{\pm 38}$  on the VAX) and the scale length of the grid can be changed from microns to meters.

The PFF software currently supports up to 10 simultaneously open files with an unlimited number of datasets per file. Two pointers are used to assist the user in managing these files: a current file pointer and a dataset pointer. The current file pointer indicates which of the open files is the current file. If no file is specified, PFF software assumes that the operation is to be performed on the current file. To simplify file handling, a number is associated with each file which is opened. This number, or file id ranges from 1 to 10 if it is assigned by the PFF program. The user may specify any positive integer as the file id as an alternative to the default value. The dataset pointer indicates the current dataset in each file. When a file is opened the dataset pointer is set to the first dataset (1) and is automatically advanced to the next dataset after a read or a write. If no dataset is specified, PFF software assumes that the operation is to be performed on the current dataset, except for a write command where the new dataset is **always** appended to the end of the file.

## 1.5 Getting Started

To run IDL on the HP lan, simply type:

```
idl
```

If you have additional procedures which you want to be available, IDL\_PATH can be defined to include not only the IDL library but also a user library. For example:

```
IDL_PATH=+/usr/local/lib/idl/lib:+/users/lpmix/idl/lib ; export IDL_PATH
```

however if only PFIDL and IDL procedures are used IDL\_PATH will default to the proper path.

To run IDL on a different lan some environment variables must be set. A script has been written to run IDL on the SUN TPA LAN which first sets environment variables and then sets an alias to run the executable (comments begin with #):

```
# make changes from local idl to idl on hp lan and add a user library path
setenv IDL_DIR /hp/usr/local/lib/idl
# To set the path for a user library use a command as shown below
# where "+/remote/abfiluk/IDL" is replaced by the proper path.
# If standard IDL and PFIDL are the only libraries used this line can be omitted
setenv IDL_PATH +"$IDL_DIR"/lib:+/remote/abfiluk/IDL
```

```
# alias the command idl to the proper executable
alias idl $IDL_DIR/bin/idl
```

where `/hp/usr/local/lib/idl` is the path to the IDL directory and the PFIDL routines are located in a subdirectory under `/lib/`.

The environment variable `PFF_EXE` may be set to a shared library for special functions. This library must include the PFF routines.

## 1.6 PFIDL Analysis

Field, particle, grid and conductor data are stored in special data structures which are outlined in Appendix A. The general procedure for the analysis of PFF datasets is indicated below.

The file containing the datasets must first be opened. If the data is in an ASCII file, the IDL command, ***openr***, can be used; PFF files require the command, ***openpff***. The data is then read into either a waveform data format array (time history data) with a ***re*** or ***reasc*** command, or into a structure array with a ***readstr*** command. Plotting the data is accomplished with the ***plo*** or ***plotstr*** commands. Examples are illustrated in the next section.

Currently only waveform data can be written directly and easily to a file; however the ***savepffidl*** and ***restorepffidl*** commands can be used to save the contents of current PFIDL arrays.

## 1.7 Sample First Session

First a file must be opened. Use the command:

```
openpff
```

A menu will appear and you can pick the file which you desire. Unlike some other menus, a single mouse click will select an item. When the file is opened, enter the command:

```
dir
```

This will provide a listing of the file contents in a text window. Look over the data and select the data which you wish to analyze. To look at a subset of the data, the command:

```
dir, 'string'
```

To read a waveform dataset use the command:

```
re, 1, 5 {or} re, 1, 'string'
```

where 1 is a WDF array in PFIDL and 5 is a dataset number corresponding to the desired data. As an alternative a string may be used to specify the dataset. To see the data use the command:

```
plo, 1
```

where 1 is the PFIDL array number. Multiple plots can be achieved with commands of the form:

```
plo, 1, 5, /ov ; plot arrays 1 to 5 in an overlay arrangement
```

```
plo, 1, 5 ; plot arrays 1 to 5, one to a page
```

```
plo, 1, 5, g = 5 ; plot arrays 1 to 5 with 5 to a page
```

**plo, [1,3, 5, 7, 9], /ov ; plot arrays 1, 3, 5, 7, and 9 in an overlay plot**  
where the semicolon indicates the beginning of a comment.

## 2.0 Functional Command Summary

### Filter Routines

- CONV -- Convolve a signal with a response function using FFT's
- DECON -- Deconvolve the source signal from the composite signal and the response function using FFT's
- FILT -- Fourier filter routine for band pass, band reject, high pass, and low pass filters. The following routines are called by FILT:
  - BAP -- Band pass filter for an array
  - BAR -- Band reject filter for an array
  - HIP -- High pass filter an array
  - LOP -- Low pass filter an array
- GET\_RESP -- Calculate the response of a detector from the composite signal and the source signal
- MED -- Median type low pass filter
- SMO -- Boxcar type low pass filter

### Help

- LHELP -- Provide a summary of all WAAV routines

### Math Routines

- ADD -- Add two arrays or an array and a constant
- AVE -- Calculate the average value of a waveform array  
Note: SUM calculates the sum or average of multiple WDF arrays.
- DIF -- Take first derivative of an array (differentiate)
- DIV -- Divide two arrays or an array by a constant
- ENVELOPE -- Calculate the envelope of a rapidly varying function
- FFTF -- Fast Fourier transform of an array
- FWHM -- Function to return the full width at half maximum
- IDL Functions -- Performs standard IDL functions on an array; these include absolute value, arc-cosine, natural and base 10 logarithm, arc-sine, arc-tangent with 1 or 2 arguments, hyperbolic sine, cosine and tangent, exponential function, square root, tangent, real and imaginary parts of a complex array, forward

and inverse Fourier transform, maximum, minimum, total, complex conjugate, I,J, and Y Bessel functions, error functions, gamma functions, statistical functions and others.

- INT -- Integrate an array
- MIRROR -- Mirror a waveform about a point
- MN -- Function to return minimum value and time of minimum
- MUL -- Multiply two arrays or an array and a constant
- MX -- Function to return maximum value and time of maximum
- RMBASE -- Subtract a background noise pulse and time shift data to a common reference
- RTP -- Raise an array to a power
- SUB -- Subtract two arrays or an array and a constant
- SUM -- Calculate the total or average of WDF arrays

### Miscellaneous

- ENDPOST -- Close a postscript file
- PS2X -- Switch to X-window output from postscript
- READASC -- Read column(s) of data from a text file
- STARTPOST -- Open a postscript file
- STARTEPS -- Open an encapsulated postscript file
- X2PS -- Switch to postscript output from X-window output

### PFF Utilities

- CLOSEPFF -- Close a PFF file
- CREATEPFF -- Create a PFF file
- DIRPFF -- List a directory of datasets in a PFF file
- ENDPFF -- End PFF file activities and close all PFF files
- OPENPFF -- Open an existing PFF file
- READHDR -- Read the header information for a PFF dataset
- READPFF -- Read one block of a PFF dataset into IDL arrays
- SETPFF -- Set the default PFF file
- SHOWPFF -- Show all opened PFF files and their file id's
- SP2XYZ -- Convert packed spatial array to individual arrays
- WRITEPFF -- Write IDL array(s) to a PFF file

**QuickSilver / Image Analysis**

- ADDSTR -- Add two structure arrays
- DELSTR - Delete a structure array
- GET\_ARRAY-- Return an array and label from a structure
- IMAGSCALE-- Interpolate an image to a different size
- PLOTFLD -- Plot field data - Multiple Block Data
- PLOTPAR -- Plot particle data - Single Block Data
- PLOTSTR -- Plot structure array
- PLOTWT -- Plot particles with specified color weighting
- READGEO -- Read geometry data
- READGRD -- Read grid data
- READFLD -- Read field type data (Fields, charge density, ...)
- READIFL -- Read an integer/float list dataset type
- READPAR -- Read particle data
- READSTR -- Read structure array

**WDF Input/Output**

- PLO -- Plot single or multiple WDF arrays
- PRIAR -- Print parameters of an array (maximum, minimum, number of points, abscissa spacing, axis labels, ordinate values,...)
- RE -- Read a PFF dataset into a WDF array
- WHELP -- Print a directory of the WDF arrays
- WRI -- Write a WDF array to a PFF file
- XPLO -- Plot one WDF array versus a second WDF array at common abscissa values

**WDF Utilities**

- CHA -- Change parameters of an array
- COM -- Least square fit of two arrays
- DELWDF -- Clear the data in WDF arrays
- GET\_MAXWDF -- Return the maximum number of WDF arrays
- GET\_VALUE-- Function to return a WDF array value at a given abscissa value

GETF	-- Get the x values for the FFT of a WDF array
GETX	-- Get the x values of a WDF array
GETY	-- Get the y values of a WDF array
GETYF	-- Function form of GETY
I2W	-- Convert two IDL arrays to a WDF array
IDL2WDF	-- Convert two IDL arrays to a WDF array
LAB	-- Label an array (CHA can also be used)
LIM	-- Limit the extent of an array or group of arrays
NEWEND	-- Add zeros to the end of an array
PUTX	-- Put a x array into a WDF array
PUTY	-- Put a y array into a WDF array
TSH	-- Time shift an array
W2I	-- Convert a WDF array to an IDL array
WDF2IDL	-- Convert a WDF array to an IDL array
WEDIT	-- Edit points in a WDF array
WDFIT	-- Fit a polynomial to a WDF array
WIN	-- Window an array using a cursor
XFR	-- Copy an array to another array

## 3.0 Examples of PFIDL

### 3.1 Read waveform arrays and plot the waveforms

Read all the waveforms in a file, 'trl5591' with the string 'mocam' and plot all the waveforms on one page on separate grids and all on one page with an overlay plot.

```
openpff, 'trl5591' ; open the file
re, 1, 'mocam', /all, n=n ; read beginning with WDF array 1 all the desired data
plo, 1, n, g = n ; plot n curves to a page, N is returned by the read command
plo, 1, n, /ov ; make an overlay plot with all the curves
```

### 3.2 Calculate power from current and voltage.

Read all the waveforms with volt and curr and average the values. Calculate an average of the currents and voltages. Calculate a baseline from the waveforms before 40 ns and subtract. Calculate the power, pulse width and total energy.

```
re, 1, 'volt', /all, n= nv ; read voltages
re, 10, 'curr', /all, n= nc ; read currents
sum, 1, nv, clabel = 'Average volt', 21
sum, 10, nc, clabel = 'Average current', 22
; Calculate baseline
vave = ave( 21, left = -100e-9, rig= 40e-9)
cave = ave( 22, left = -100e-9, rig= 40e-9)
; Subtract baseline
sub, 21, 0, vave
sub, 22, 0, cave
; Calculate power
mul, 21, 22, 23, clab = 'Power'
plo, 23 ; plot the power
print, 'Power pulse (ns) = ', 1e9*fwhm(b=0, /q, 23)
int, 23, 24, clab = 'Energy'
print, 'Total energy = ', mx(24)
plo, 23, /left , tit = 'Power and Energy'
plo, 24, /right
```

### 3.3 Generate magnetic stream function

Generate a magnetic field stream function from a magnetic field and an initial value of the stream function.

```
openpff, 'qsnp'
readstr, 1, 6 ; read initial stream function into structure 1
readstr, 2, 3, 3, /con ; read the conductors into structure 2
```

```

border, 1, 1, 10 ; store the values of the stream function for minimum first
                  ; dimension in WDF array 10
readstr, 3, 158 ; read magnet field data into structure 3
plotstr, 3, 3, 2, [0, 1], work = 4, con = 2 ; Plot the 3 component of the field averaged
                  ; over the range [0, 1] in the second dimension and store the image
                  ; in structure array 4 and plot the conductor
plotstr, 4, /xint = 2, work = 5, start = 10, con = 2 ; calculate the integral  $\int r dr$  of array
                  ; 4 and add the values in waveform array 10 and plot the result
                  ; with conductors

```

### 3.4 View a scanned CR-39 Image

Read a CR-39 scanned image, plot the individual images and plot a composite image

```

openpff, 'qc5851'
readstr, 1
!p.multi = [0, 4, 5]
for i = 1, 20 do plotstr, 1, 0, 3, i, /in
!p.multi = 0
window, 1
plotstr, 1, 0, 3, [3, 12], /index, /fill

```

### 3.5 Plot lineouts from an image file

Read an image file, plot the images at each time and plot lineouts. Also print the full width at half maximum of the images as a function of time.

```

openpff, 'movie.pff' ; open pff file with the image
readstr, 1
time = get_array(1, 3)
yval = get_array(1, 2)
ny = n_elements(yval)
ymid = total(yval)/ny
ydel = (max(yval, min=mn)-mn)/(ny-1)
ylo = ymid -ydel & yhi = ymid+ydel
nt = n_elements(time)
for i = 1, nt do begin
    plotstr, 1, 1, 3, i, /index, tit = 'Image at t =' + string(time(i-1)) , $
    /xline, wid = [ylo, yhi], out = i
endfor
plo, 1, nt, /ov, /color

```

### 3.6 Waveform analysis

Read signals from 4 different files scale and plot on a common grid; time shift the data, and plot on a common grid

```

Pro voltage
openpff,'file1', 11      ; Open the four files
openpff,'file2', 12      ; Note: File id's of 11 through 14 are specified
openpff,'file3', 13      ;      to insure unique file id's for these files.
openpff,'file4', 14
; Make an array and define the time shifts for each dataset
ts= - [430, 219, 365, 233]*1.0e-9
for j=1,4 do re, j,'Voltage', fi= j+10 ; Read the dataset with 'Voltage' in the comment
for j = 1, 4 do tsh, j, ts(j-1)      ; Time shift the data
tsh, 1+ indgen(4)      ; Make certain all arrays begin at 0
lab, 1, 'File 1'      ; Label the datasets
lab, 2, 'File 2'
lab, 3, 'File 3'
lab, 4, 'File 4'
; Change the time base to nanoseconds
cha, 1+indgen(4), x = 'Time (ns)', m = 1e9
; Make an overlay plot of the four signals with a legend in color
plo, 1, 4, /ov, /color, title = 'Voltages from 4 files'
hak, /mes      ; pause until a key is hit
plo, 1, 4, g= [2,2]      ; Plot the 4 waveforms on individual grids with 4 to a page
return
end

```

### 3.7 Command Procedure for QUICKSILVER Post-Processing

The procedure below was used to plot time history results of a Quicksilver simulation with files directly as written.

```

Pro qsplotall; Procedure to plot QS output
; open the geometry file
openpff, '/usr/local/lib/idl/lib/contrib/example/qsgeo'
; make a directory with 10 entries visible and a scroll bar
dir, page = 10
readstr, 2, 2, /grid      ; read the grid into structure 2
; read the conductor in dataset 3 and following for a total of 6 into str 3
readstr, 3, 3, 6, /con
; plot the conductors (in structure 3) in color 4
; plot axis 3 for the x axis, axis 1 for the y axis and take a value of
;      zero for the missing axis (axis 2)

```

```

    plotcon, 3, 3, 1, 0.0, color = 4
; and overlay the grid data in structure 2; use colors to designate grid
; lines 0, 10, ... and 5, 15, ...
    plotgrd, 2, /over, color = [1, 2,2,2,2, 7, 2,2,2,2]
; another way to do the same thing is given below; the conductors are white
    plotgrd, 2, 3, 1, 0.0, color = [1, 2,2,2,2, 7, 2,2,2,2], con = 3
; Pick opposite corners of a square and zoom the grid in another window
    print, ' Pick one corner of the zoom window'
    cursor, x1, y1
    print, ' Pick the other corner of the zoom window'
    cursor, x2, y2
    xr = [min([x1, x2], max = mx), mx]
    yr = [min([y1, y2], max = mx), mx]
    window, 1
    plotgrd, 2, 3, 1, 0.0, color=[1,2,2,2,2,7,2,2,2,2], con=3, xr=xr, yr=yr
; open the snapshot file
    openpff, '/usr/local/lib/idl/lib/contrib/example/qssnp'
; make a directory of the snapshot file in another window
    dir, page = 15, /res
; set up a loop to look at the electric field in a y slice above and
; below the dielectric
; first read the first y slice (dataset 2) into structure 5
    readstr, 5, 2
; use shelp to see the components of the structure 5,
; Note: 2 y components at Xj- Min = -0.000330200, Max = 1.45519e-11
; plot the vacuum component in the top and the dielectric component in the
; bottom of the window.
    window, xsize = 700, ysize = 800
    !p.multi = [0, 1, 2]
    for i = 2, 600, 6 do begin
        readstr, 5, i
        plotfld, 5, [1,2,3], 2,1, /in,/tr, /fill, con= 3, tit = 'Dielectric'
        plotfld, 5, [1,2,3], 2,2, /in,/tr, /fill, con= 3, tit = 'Vacuum'
        empty
        hak, /mes
    endfor
; plot again with levels set for all plots - divide levels
; for the dielectric by 5
    lev = findgen(59)*120.
    lev5 = lev/5.0

```

```

for i = 2, 600, 6 do begin
  readstr, 5, i
  plotfld, 5, [1,2,3], 2,1, /in,/tr, /fill,con=3,lev=lev5,tit='Dielectric'
  plotfld, 5, [1,2,3], 2,2, /in,/tr, /fill,con=3,lev=lev,tit='Vacuum'
  empty
  hak, /mes
endfor
; open the time history file
openpff, 'usr/local/lib/idl/lib/contrib/example/qshis'
; make a directory
dir , page = 10, xsize = 300, /reset
; read all the datasets with 'v' into waveform arrays 1, ...
re, 1, 'v', /all
!p.multi = [0, 3, 1]
; plot in groups of three with 3 grids to a page
plo, 1, 3, /ov, /color
plo, 4, 3, /color, /over
plo, 7, 3, /color, /over
; print the fwhm of datasets 2, 5, and 8 with a baseline of zero
for i = 2, 8, 3 do print, fwhm (i, b= 0)
; reset the plot counter
!p.multi = 0
; plot all 9 curves on one page
plo, 1, 9, g= [3,3]
return
end

```

### 3.8 Command Procedure for QUICKSILVER Post-Processing

The procedure below was used to plot contour of the magnitude of the electric field with an overlay of the electric field.

```

Pro plotall           ; procedure to plot QS output
openpff, 'qsnp'
dir                   ; print a directory of the file
readstr, 1, 2, 3, /con ; read the conductor datasets from 2,3,4
readstr, 2, 16        ; read dataset 16
plotfld,2, con= 1, [1,2,3], 2, [0.0, !pi/3] ; plot the magnitude of the fields from
;
;                   dataset 16 with the conductors in array overplotted
;                   average the second dimension between zero and pi/3
return
end

```



## 4.0 Command Reference Dictionary

Each of the PFIDL commands is discussed in alphabetical order in this section with examples. All arguments in [ ] are optional.

### ACTIVATE

Calculate activation yield from voltage and current waveforms. Current and voltage will be put on a uniform time base using linear interpolation. (Hooks are there to use xy type data) Cross sections have been provided by Gary Cooper.

format: Response = ACTIVATE(WDV, WDI [, WDOUT] [, REACTION = reaction] [, CHARGE = charge] [, PRINT = print])

where:

- WDV* - Waveform data array number for the voltage, units = Volts
- WDI* - Waveform data array number for the current, units = Ampere. Negative values of current are set to zero.
- WDOUT* - Waveform data array number for the response  
Default is (Maxwdfarray-1)

Keywords:

- CHARGE* - Ion charge. Default = 1  
Voltage will be multiplies by this number and current divided by this number to obtain particle energy and current.
- REACTION* - Desired reaction indicated by a number given by the following:
  1. Li on LiF -> 24Na
  2. H on LiF -> 7 Be
  3. H on BN -> 7 Be
  4. C on BN -> 24 NaIf not specified a widget will be provided for the selection.
- PRINT* Print the yield curve values  
**Note:** These values are also stored in Maxwdfarray and may be plotted with: Plo, get\_maxwdf()

Outputs:

The number of joules/Bq for the given current, voltage and reaction is returned. The response as a function of time and energy are in WDOUT ( or Maxwdfarray-1) and Maxwdfarray.

**Note:** If all ions are below threshold then zero is returned.

**Procedure:**

Limit voltage to > 5% of maximum. Adjust current and voltage for charge if not 1 calculate current yield for these voltages. Integrate yield and energy to obtain response (joules/Bq)

**Examples:**

Assume counts = [28,20,30] is the activation of 3 shakers. Calculate the energy/cm<sup>2</sup> for each if the area of each is .02 cm<sup>2</sup>. Assume voltage in array 2 and current in array 3 and the ions have a charge of 1 and the reaction is protons of LiF.

```
ener = ACTIVATE (2,3, r=2)/.02*counts
print, 'Shaker Counts Energy/cm2'
for i=0, 2 do print, i, counts(i), ener(i)
```

Same as the last example but assume the protons go through a 20 micron aluminum filter(z=13) before hitting the LiF.

```
vout = -9 ;set the unknown quantity as a variable
stopping, /quiet, 1, getyf(2)*1e-6, vout, 20, 13 ; Note: the 1 is protons, voltage is
scaled to MeV, 20 & 13 are the foil
puty, vout*1e-6, 2 ; return the attenuated voltage to wdf array 2 ; the rest is the
same as before:
ener = activate(2,3,r=2)/.02*counts
print, 'Shaker Counts Energy/cm2'
for i=0, 2 do print, i, counts(i), ener(i)
```

**ADD**

Add a WDF array to another WDF array at corresponding abscissa values. Linear interpolation is used to put the arrays on the same abscissa values. The result only contains values in the region of overlap of the two arrays.

**Note:** *TSH* and *NEWEND* can be used to add zero values to the beginning and end of a WDF array.

**Note:** If *wd3* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

```
format:  ADD, wd1, wd2 [, wd3] [, CLABEL = clabel] [, XLABEL = xlabel]
        [, YLABEL = ylabel]
```

where:

<i>wd1</i> , <i>wd2</i> , <i>wd3</i>	- WDF array numbers If <i>wd3</i> is present, then $wd3 = wd1 + wd2$ else $wd1 = wd1 + wd2$
<i>CLABEL</i>	- Dataset comment for the sum Default is the dataset comment for <i>wd1</i>
<i>XLABEL</i>	- X-axis label for the sum Default is the x-axis label for <i>wd1</i>

*YLABEL* - Y-axis label for the sum  
Default is the y-axis label for *wd1*

Examples:

Add array 1 to array 2 and store the result in array 3

```
ADD, 1, 2, 3
```

Add array 3 to array 4 and store the result in array 3. Label the sum 'Total Power'

```
ADD, 3, 4, c = 'Total Power'
```

Add a constant to each element of a WDF array.

**Note:** To add two constants, use IDL directly. (See last example.)

format: **ADD, wd1, 0, value [, wd3]**

where:

*value* - constant or first element of an array

*wd1, wd3* - WDF array numbers

If *wd3* is present, then  $wd3 = wd1 + value(0)$   
else  $wd1 = wd1 + value(0)$

Examples:

Add 10 to all the elements in array 1 and store the result in array 3

```
ADD, 1, 0, 10, 3
```

Add the variable, *shift*, to all the elements in array 3 and store the result in array 3

```
ADD, 3, 0, shift
```

Add array 4 to arrays 1, 2, 3 and store the results in arrays 5, 6, 7

```
for i=1, 3 do ADD, i, 4, 4+i
```

Add variable, *girls*, to variable, *boys*, and store in *total*

```
total = boys + girls
```

```
plo, 2
```

## ADD\_FRAMES

Add frames in a CR-39 digitized image. See the function ADDFR

```
plo, 2
```

## ADDFR

Add the frames in a CR-39 digitized image. Input structure must be a single block, 3 dimensions. Output structure will be 2 dimensions.

format: **STR\_TOTAL = ADDFR(STR\_IN, /ALLFRAMES)**

format: **STR\_TOTAL = ADDFR(STR\_IN, FRAMES)**

**format: STR\_TOTAL = ADDFR(STR\_IN, FRAMES [, NFRAMES])**

where:

- STR\_TOTAL* - Composite image equal to the sum of the designated frames
- STR\_IN* - Structure or structure array number with the CR-39 images. This may be any 3 dimensional, single block structure
- ALLFRAMES* - If set, frames 1 to 19 will be added. If the number of frames (size of dimension 3) is not equal to 20, then the frames from 1 to (size of dimension 3)-1 will be added. If ALLFRAMES is set, all other parameters are ignored.
- FRAMES* - Array of frames to be added. These values correspond to indices in the third dimension. For current data these are in the range of 1 to 20. If FRAMES may be a single value.
- NFRAMES* - Number of frames to be added. If FRAMES is an array, NFRAMES is ignored. Default value = 1

Outputs:

Add structure arrays 2 and 3 and store the result in structure 8

Background:

CR-39 images are digitized according to the number of tracks with a given size in a bin. Low resolution scans are 300 microns/bin. High resolution scans are 150 microns/bin.

Frame Number	Track Area (Pixels = 0.299 $\mu\text{m}^2$ ) /	Effective Diam
1	1 to 2 pixels	/ 0.617 to 0.872 microns
2	3 to 5 pixels	/ 1.069 to 1.380 microns
3	6 to 10 pixels	/ 1.511 to 1.951 microns
4	11 to 16 pixels	/ 2.046 to 2.468 microns
5	17 to 23 pixels	/ 2.544 to 2.959 microns
6	24 to 32 pixels	/ 3.023 to 3.490 microns
7	33 to 42 pixels	/ 3.544 to 3.999 microns
8	43 to 53 pixels	/ 4.046 to 4.492 microns
9	54 to 65 pixels	/ 4.534 to 4.974 microns
10	66 to 94 pixels	/ 5.013 to 5.982 microns
11	95 to 128 pixels	/ 6.014 to 6.981 microns
12	129 to 168 pixels	/ 7.008 to 7.997 microns
13	169 to 212 pixels	/ 8.021 to 8.984 microns
14	213 to 262 pixels	/ 9.005 to 9.987 microns
15	263 to 378 pixels	/ 10.006 to 11.996 microns

16	379 to 514 pixels	/	12.012 to 13.989 microns
17	515 to 672 pixels	/	14.002 to 15.995 microns
18	673 to 851 pixels	/	16.007 to 17.999 microns
19	852 to 9999 pixels	/	18.010 to 112.832 microns
20	Saturation Frame - Total Number of Pixels with tracks normalized to the number of pixels in the bin times 32768. (a value of 16384 means half of the pixels contain tracks)		

**Example:**

Plot the composite of all frames in structure 1 except the saturation frame.

```
plotstr, ADDFR(1, /all)
```

Plot the composite of structure 1 frames 3, 4, 5 and 6

```
plotstr, ADDFR(1, 3, 4)
```

or

```
plotstr, ADDFR(1, [3, 4, 5, 6])
```

Store the composite of structure 1 frames 3, 4, 5, and 6 in structure dog

```
dog = ADDFR(1, 3, 4)
```

Store the composite of structure 1 frames 3, 4, 5, and 6 in structure array 2

```
i2s, ADDFR(1,3, 4), 2
```

Plot frame 5 of structure 2

```
plotstr, ADDFR(2, 5) ;nframes has a default value of 1
```

or

```
plotstr, ADDFR(2, 5, 1)
```

or

```
plotstr, 2, 3, 5, /index
```

**ADDSTR**

Add a field structure array to a field structure array or to a constant. Store the result in the initial field structure array or another structure array. All attribute or vector quantities are added.

```
format: ADDSTR, STR1, STR2, [, STR3] [, CLABEL = clabel] [, MASTER = master]
        [, CUBIC = cubic]
```

```
format: ADDSTR, STR1, 0, VAR, [, STR3] [, CLABEL = clabel]
```

where:

*STR1* - array number of the first field structure array

*STR2* - array number of field structure to be added to first structure

or

*0, VAR* - Variable or constant to be added to the first

- structure
- STR3* - array number for the sum, if not specified the sum is stored in STR1 (the first array)
- CLABEL* - Optional label for the combined structures.
- MASTER* - Optional grid specification for structures with different grid specifications. Single block data is normally put onto a uniform grid using the minimum spacing in STR1 or STR2 for each coordinate. Multiple block data is normally not handled if the grids specifications are different. If MASTER is specified, then the result of the ADDSTR command will have the same grid as STR1 or STR2 depending on the value of MASTER (1 or 2). For single block data, if MASTER is -1 or -2 then the grid will be truncated to the region of overlap with the other structure.
- CUBIC* - If set a cubic interpolation will be used to match data on different grids

**Restrictions:**

STR1 array must be defined and have valid data  
STR2 or 0 and a Variable must be defined.

If MASTER is specified, there are no other restrictions

STR1 and STR2 must both be field arrays and have the same number of vector components.

The SIZE of STR1 and STR2 must be the same

The values of the spatial coordinates must be equal to about  $3e-5$  (ie.  $-\max(\text{abs}(\text{difference}))/\max(\text{value}) \leq 3.1e-5$ )

The check on spatial coordinates is done globally rather than on a block by block basis.

If a structure has three dimensions, but is degenerate in one of the dimensions that axis need not be equal.

**PROCEDURE:**

If STR2 is specified:

Each attribute, or vector component, of STR1 is added to each attribute of STR2

If 0, VAR is specified:

VAR is added to each attribute of STR1

In either case, the sum is stored in STR3, if specified, or STR1, if STR3 is not specified.

**EXAMPLES:**

Add structure arrays 2 and 3 and store the result in structure 8

**ADDSTR, 2, 3, 8**

Add the variable, OFFSET, to structure array 2 and store the sum in structure 8

**ADDSTR, 2, 0, offset, 8**

Add structure arrays 3 and 8 and store the result in structure 10 with a dataset comment, 'Total Charge - Line + Diode'

**ADDSTR, 3, 8, 10, c= 'Total Charge - Line + Diode'**

Add structure array 1 to structure array 2 and store the result in array 3; insure the result has the same spatial grid as array 1

**ADDSTR, 1, 2, 3, /m**

or

**ADDSTR, 1, 2, 3, master = 1**

## ADD\_COYOTE

Add the coyote directory to a IDL session path. Caution: Some routines in the coyote directory conflict with the IDL distribution.

format: **ADD\_COYOTE]**

## ARR2STR

Convert a two or three dimensional array to a field type structure array with one or more vector components.

format: **ARR2STR, str, image, image2, image3, FEMesh = femesh, PLOTIT = plotit, x, y, z, CLABEL = clabel, XLABEL = xlabel, YLABEL = ylabel, ZLABEL = zlabel, TLABEL = tlabel, FILE = file, V1LABEL = v1label, V2LABEL = v2label, V3LABEL = v3label**

where:

- str* - Structure array number
- image* - 2D or 3D array
- image2, image3* - Additional images for a Vector type field structure  
**Note:** Image, image2 and image3 must all be the same size
- x* - Vector of x values, must be as big as first dimension
- y* - Vector of y values, must be as big as second dimension
- z* - Vector of z values, must be as big as third dimension

Keyword:

- FEMesh* - If set, data is from a finite element type, random mesh. Default: FEMesh = 0; rectilinear mesh  
If set, X, Y are required and only 1 image is allowed. Image, x, and y are linear arrays.
- PLOTIT* - Ignored unless FEMesh is set. Plots the triangles.
- CLABEL* - Comment label

<i>TLABEL</i>	- Type label
<i>XLABEL</i>	- X-label
<i>YLABEL</i>	- Y-label
<i>ZLABEL</i>	- Z-label
<i>V1LABEL</i>	- Vector 1 label
<i>V2LABEL</i>	- Vector 2 label
<i>V3LABEL</i>	- Vector 3 label
<i>FILE</i>	- Associated file

**Examples:**

Make a structure array from array, picture, and put it into 4

```
ARR2STR, 4, picture, clab = 'filtered image'
```

Make a structure array from arrays, vx, vy and vz with spatial coordinates, xval, yval, zval and store into structure 18

```
ARR2STR, 18, vx, vy, vz, xval, yval, zval, clab = 'Particel velocity'
```

**AVE**

Function which returns the average value of a waveform data array or a portion of the array.

**Note:** The IDL function TOTAL is used to calculate the average.

**Note:** Use SUM, /AVERAGE to obtain the average of a number of waveform arrays.

**format:** **AVERAGE = AVE(wd1 [, SD = sd] [, LEFT = left] [, RIGHT = right] [, QUIET = quiet])**

where:

<i>AVERAGE</i>	- average value of the WDF array
<i>wd1</i>	- WDF array number
<i>sd</i>	- Standard deviation = $\sqrt{\text{total}(y-\text{ave})^2 / (n-1)}$
<i>LEFT, RIGHT</i>	- optional abscissa value to specify the left and right abscissa values used to determine the local average value of the waveform. If absent, the default values are the first and last data points, respectively.
<i>QUIET</i>	- if present and nonzero, no values will be printed to the terminal

**Examples:**

Print the average value of array 2 between 10 and 1000

```
PRINT, AVE(2, l = 10, r = 1000)
```

Eliminate a baseline offset in waveform 2 by subtracting the average value of the baseline up to the time the signal starts at 35ns

```
sub, 2, 0, AVE(2, r= 35e-9)
```

Calculate the average value of array 3 and store it in average; suppress terminal output

```
AVERAGE = AVE(2, /quiet)
```

## BAP

Apply a band pass filter to a WDF array.

**Note:** If *wdf2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wdf1*.

**Note:** See *FILT* for keyword definitions. This routine is called by *FILT* and is not recommended for general use; use “*filt*, /*bp*”.

```
format:  BAP, wdf1 [, wdf2] [, TYPE = type] [, ATTEN = filter] [, ORDER = order]
         [, MODE= bin] [, WIDTH = width] [, /PLOT].
```

where:

*wd1*, *wd2* - WDF array numbers  
 If *wd2* is present, then *wd2* = BAP(*wd1*)  
 else *wd1* = BAP(*wd1*)

Default keyword values are:

```
type = 'BUT'
order =2
bin = 12
width = 0.8*bin
```

Examples:

Apply a band pass filter to WDF array 1 and store the filtered array in array 2 using the default parameters

```
BAP, 1, 2
```

## BAR

Apply a band reject filter to a WDF array.

**Note:** If *wdf2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wdf1*.

**Note:** See *FILT* for keyword definitions. This routine is called by *FILT* and is not recommended for general use; use “*filt*, /*br*”.

```
format:  BAR, wdf1 [, wdf2] [, TYPE = type] [, ATTEN = filter] [, ORDER = order]
         [, MODE = bin] [, WIDTH = width] [, /PLOT].
```

where:

*wd1*, *wd2* - WDF array numbers  
 If *wd2* is present, then *wd2* = BAR(*wd1*)  
 else *wd1* = BAR(*wd1*)

Default keyword values are:

```
type = 'BUT'
order =2
```

*bin* = 12  
*width* = 0.8\**bin*

**Examples:**

Apply a band reject filter to WDF array 1 and store the filtered array in array 2 using the default parameters

**BAR, 1, 2**

**BLINE**

Subtract a constant value from a waveform. Baseline can be set by clicking left mouse button or by entering a value. Once a baseline is set the baseline will follow the cursor. Baseline is accepted with the middle mouse button. Program exits. Selected baseline is reset with the right mouse button. Exit/Quit button will exit with no baseline.

**format:** **BLINE**, *wdf1* [, *wdf2*] [, **OFFSET** = *offset*] [, **XRANGE** = *xrange*] [, **/NO\_SAVE**] [, **XSIZE** = *xsize*] [, **YSIZE** = *ysize*] [, **\_EXTRA** = *extrastuff*]

where:

- wdf1* - WDF array number or an array name
- wdf2* - WDF array number to store the result
- Default WDF1 = WDF2
- OFFSET** - value of the baseline selected. Default = 0.0 If quit is selected, offset = 0.0
- XRANGE** - keyword passed to the PLO command
- NO\_SAVE** - if set, only value will be returned
- XSIZE** - size in pixels for the draw widget
- YSIZE** - size in pixels for the draw widget
- Default values set in set\_drawsize
- \_EXTRA** - any other valid keywords to the plo command

**Example:**

Select the baseline on waveform 4 and subtract it.

**BLINE, 4**

Select the baseline on waveform 5 and the result in waveform 15. Plot only from -100 to 50 ns

**BLINE, 5, 15, xr = [-100, 50]\*1e-9**

**BORDER**

This routine calculates the values along the minimum or maximum dimension of a structure array. The structure array must be two dimensional or be three dimensional with a degenerate dimension and a single attribute.

**format:** **BORDER**, **STR**, **SIDE**, **WDA**, **TOP** = *top*, **PLOT** = *plot*

where:

- STR* - Field structure or a QS structure array number
- SIDE* - Indicates the coordinate for which the minimum or maximum is desired
- WDA* - Waveform array to store the values
- TOP* - If present and not zero, the structure values at the maximum values of dimension, *side*, will be calculated. The default is the minimum values of dimension, *side*.
- PLOT* - If present and not zero, a plot of the values will be made

Example:

Store in WDF array 34, the value at minimum r for the second field component of structure 2. Average over the second spatial dimension from -1 to 10.

```
tempstr = slice(2, 2, 2, [-1, 10])
BORDER, tempstr, 1, 34
```

## BOX

Draw a box grid with different labels on all four sides. Scaling will be according to the x1 and y1 values unless save is set.

format: **BOX**

where:

- X1RANGE* - Xrange for bottom axis.
- X2RANGE* - Xrange for top axis.
- Y1RANGE* - Yrange for left axis.
- Y2RANGE* - Yrange for right axis.
- X1TITLE* - Xtitle for bottom axis.
- X2TITLE* - Xtitle for top axis.
- Y1TITLE* - Ytitle for left axis.
- Y2TITLE* - Ytitle for right axis.
- SAVE* - If set, plot parameters will be set to the second set of axes. (X2range, Y2range)
- \_EXTRA* - Any valid keywords for the plot command. (ticklen, margin, etc)

Parameters set by box command are:

```
NoData = 1
xstyle, ystyle = 9
xmargin, ymargin = !x.margin(0), !y.margin(0)
```

**CHA**

Change the parameters of a WDF array or an array of WDF arrays.

**format:** CHA, wda [, CLABEL = clab] [, YLABEL = ylabel] [, XLABEL = xlab]  
 [, SCALE = scale] [, NPOINTS = np] [, TZERO = tzero] [, DELTAT = dt]  
 [, ADT = adt] [, MDT = mdt]

where:

- wda* - WDF array number or an array of numbers
- CLABEL* - dataset comment (used for title of plots)
- YLABEL* - y-axis (ordinate) or block label
- XLABEL* - x-axis (abscissa) label
- append* - If set, append the string in CLABEL, XLABEL or YLABEL to the corresponding label with an intervening space unless append is a string. If append is a string, use this string as the separator.
- aclabel, xlabel, ylabel* - Same as append but only apply to the comment xlabel or ylabel respectively.
- SCALE* - multiply the amplitude by the specified scale factor. This is identical to: **mul, wda, 0, scale**  
 or  
 scale may be any valid IDL command string to set the amplitude to a function of x or y.  
 CHA uses `ok = execute(' y = '+ scale)` where x and y are the abscissa and ordinates of the wdf array.  
 example: to set the amplitude to  $\exp(x)*y$  or  $(\sin(x)/y)$  or a user specified function use:  
`scale = 'exp(x)*y' ; or`  
`scale = 'sin(x)/y' ; or`  
`scale = 'My_Function(x, y)'`  
**Note:** The user is required to insure valid data.
- NPOINTS*<sup>†</sup> - number of points is changed to this value and the interval between points changed to allow the dataset to cover the same domain
- TZERO* - specifies a start time for the dataset; this parameter is also changed by the TSH command.
- DELTAT*<sup>†</sup> - interval between points is changed to this value and the number of points changed to allow the dataset to cover the same domain.
- ADT* - specifies an absolute change in the interval between points; the number of points remains the same
- MDT* - specifies a multiplier for the current abscissa values; the initial time and interval between abscissa values are

*multiplied by this value*

**Note:** All keywords may be abbreviated by the first letter.

**Note:** If X-Y data is stored in the waveform array, it will be converted to uniformly space abscissa values if npoints, deltat, or adt is specified.

†The number of points and the interval between points cannot be changed independently with these parameters. If either is changed, linear interpolation will be used to put the array on the new time base. If the number of points is reduced or the time interval increased, then the array will be filtered with a boxcar filter with the width, in units of data values, equal to  $1.2 * (1 + \text{new deltat} / \text{old deltat})$ .

Examples:

Change the time base and axis label from seconds to nanoseconds for arrays 1,3

**CHA, [1,3], m = 1e9, x = 'Time (ns)'**

Change the comment label of array 4 to included the shot number at the end. Assume shot is in variable shot.

**Shot= 5791**

**CHA, 4, c= shot, /ac**

or

**CHA, 4, c= shot, /ap ; since only one label is changed**

Similar to the example above, but use ' - ' as a separator

**Shot= 5791**

**CHA, 4, c= shot, ac= ' - '**

or

**CHA, 4, c= shot, append = ' - '**

Change the dataset comment for array 2, scale the amplitude and x axis and change the x- and y-axis labels.

**CHA, 2, c='Energy on Target', scale=1e-3,y ='Energy (kJ)', m = 1e9,x='Time (ns)'**

Change the dataset comment for array 3 to 'Energy of the Plasma'

**CHA, 2, c='Energy of the Plasma'**

**Note:** The LAB command can be used to obtain the same result.

**lab, 3,'Energy of the Plasma'**

Change the axis labels of the graph for arrays 2 through 10 and change the abscissa increment to 1 and the initial abscissa value to 0.0

**CHA, indgen(9)+2, x= 'Time Step', y= 'Particle Flux', a= 1, tzero = 0.0**

## CHASTR

Change the parameters of a structure array

All types of structures arrays are supported

Changes which do not apply to a particular structure type are ignored

format: **CHASTR, STR**

where:

- STR* - Structure array number or an array of structure array numbers to be changed. If only one structure is specified, then it may be an IDL structure

Keywords:

- CLABEL* - Replace the dataset comment for the structures specified by *STR* with the string or string variable specified
- XLABEL* - Replace the first spatial dimension label for the specified structures with the string or string variable specified
- YLABEL* - Replace the second spatial dimension label for the specified structures with the string or string variable specified
- ZLABEL* - Replace the third spatial dimension label for the specified structures with the string or string variable specified
- TLABEL* - Replace the fourth spatial dimension label for the specified structures with the string or string variable specified
- ALABEL* - Replace all spatial dimension labels for the specified structures with the string or string variable specified
- V1LABEL* - Replace the first attribute label or the first vector label for the specified structures with the string or string variable specified
- V2LABEL* - Replace the second attribute label or the second vector label for the specified structures with the string or string variable specified
- V3LABEL* - Replace the third attribute label or the third vector label for the specified structures with the string or string variable specified
- V4LABEL* - Replace the fourth attribute label or the fourth vector label for the specified structures with the string or string variable specified
- V5LABEL* - Replace the fifth attribute label or the fifth vector label for the specified structures with the string or string variable specified
- V6LABEL* - Replace the sixth attribute label or the sixth vector

	label for the specified structures with the string or string variable specified
<i>VALABEL</i>	- Replace all attribute labels or all vector labels for the specified structures with the string or string variable specified
<i>MXDIM</i>	- Multiply the first spatial dimension by this value
<i>MYDIM</i>	- Multiply the second spatial dimension by this value
<i>MZDIM</i>	- Multiply the third spatial dimension by this value
<i>MTDIM</i>	- Multiply the fourth spatial dimension by this value
<i>MADIM</i>	- Multiply all spatial dimensions by this value
<i>AXDIM</i>	- Add this offset to the first spatial dimension
<i>AYDIM</i>	- Add this offset to the second spatial dimension
<i>AZDIM</i>	- Add this offset to the third spatial dimension
<i>ATDIM</i>	- Add this offset to the fourth spatial dimension
<i>AADIM</i>	- Add this offset to all the spatial dimensions

**Note:** Multiplication and division are performed according to:

$$X_{\text{new}} = X_{\text{old}} * \text{MXDIM} + \text{AXDIM}$$

<i>S1V</i>	- Multiply or scale the first vector or attribute component
<i>S2V</i>	- Multiply or scale the second vector or attribute component
<i>S3V</i>	- Multiply or scale the third vector or attribute component
<i>S4V</i>	- Multiply or scale the fourth vector or attribute component
<i>S5V</i>	- Multiply or scale the fifth vector or attribute component
<i>S6V</i>	- Multiply or scale the sixth vector or attribute component
<i>SCALE</i>	- Multiply all the vector amplitudes by a scale factor. This is identical to: mulstr, str, 0, scale

Examples:

Change the dataset comment to 'Stream Functions at 65 ns' on structure array 2

**CHASTR, 2, c = 'Stream Functions at 65 ns'**

Change the x and y spatial coordinates of an image by 1/magnification

**CHASTR, IMAGE,mx=1/.56, my = 1/.56,xl='Target(cm)',yl='Target(cm)'**

Change the spatial coordinates of arrays 2 through 9 to cm from meters

**CHASTR,indgen(8)+2,ma=1e2,x= 'X (cm)',y='Y (cm)',z='Z (cm)'**

**Note:**In cylindrical coordinates we would use:

CHASTR,indgen(8)+2,mx=1e2, mz = 1e2, x ='Radius(cm)',z='Z(cm)'

Change the units of electric field to MV/cm from V/m on structure, F

CHASTR, F, sa= 1./1e8,v1='Ex(MV/cm)',v2='Ey (MV/cm)',v3='Ez(MV/cm)'

## CLOSEPFF

Close a PFF file. If the current file is closed, the current file pointer will be set to the next file opened or to the previous file it is the last file.

format: **CLOSEPFF** [, fileid]

where:

- fileid* - file id of file to be closed.  
 If it is omitted, the current file is closed.  
 If it is zero or "all", then all files will be closed.

Examples:

Close the current file

**CLOSEPFF**

Close the file with a file id of 3

**CLOSEPFF, 3**

Close all files

**CLOSEPFF, 0 ; or: closepff, 'all'**

## COM

Compares two WDF arrays and attempts to find amplitude and time shift values which minimize the differences (least squares) between the arrays. The best fit of arrays, A and B, is of the form:  $A(t) = amp*B(t-tshift) + bline(1)*t + bline(0)$ . The default comparison assumes *bline(0)* and *bline(1)* are zero. Following the best fit, the left and right hand sides of this equation are plotted.

**Note:** If out is not specified, then the two maximum WDF arrays are used to store the fitted curves.

format: **COM**, wd1, wd2 [, /BASE] [, /BSLOPE] [, AMP = amp] [, BLINE = bline]  
 [, TSHIFT = tshift] [,LEG = leg] [, /NOLEG] [, /QUIET] [, OUT = out]  
 [, ERRLIM = errlim] [, INITDEL = initdel] [, \_EXTRA = extrastuff]

where:

- wd1, wd2* - WDF array numbers for the arrays to be compared  
*BASE* - nonzero value indicates a *bline(0)* will be allowed to vary from zero. This parameter is ignored if bslope is specified and nonzero.  
*BSLOPE* - nonzero value indicates a *bline(1)* and *bline(0)* will be allowed to vary from zero.

<i>AMP</i>	- returned with the calculated value of <i>amp</i>
<i>BLINE</i>	- returned with the calculated value of <i>bline</i>
<i>TSHIFT</i>	- returned with the calculated value of <i>tshift</i>
<i>LEG</i>	- a two element vector specifying the location for the best fit parameters on the plot as a fraction of the plot range. Default value is [0.04, 0.94]
<i>NOLEG</i>	- nonzero value indicates that the <b>no</b> best fit parameters will be printed on the plot. If this keyword is zero or omitted, the best fit parameters will be output to the graphics window using the values of <i>leg</i> .
<i>QUIET</i>	- nonzero value indicates that the <b>no</b> waveform comparisons will be plotted and no printing of the best fit parameters. If this keyword is zero or omitted, the best fit waveforms will be plotted and values printed on the terminal.
<i>OUT</i>	- WDF array numbers for the best fitted arrays. They are stored in the following: if out is a one element array - out and out+1 if out is a two element array - out(1) and out(2) if out is not specified or invalid - 81, 82
<i>ERRLIM</i>	- error limit for convergence; default is 3.125% of the minimum point spacing
<i>INITDEL</i>	- initial delta for the iteration. Default is 20% (> 20 dt) of the original region of overlap. If the data needs a large time shift, set <i>initdel</i> to this value. <i>Initdel</i> can be varied to look at the sensitivity of the result to the initial value.
<i>_EXTRA</i>	- any valid keywords for the PLO command

**Examples:**

Compare array 1 to array 2 with no baseline adjustment

**COM, 1, 2**

Compare array 1 to array 2 with sloping baseline and store the amplitude in *r*, the time shift in *s*, and the baseline parameters in *basepar*. Position the legend in the lower right corner.

**COM, 1, 2, /bs, a=r, t=s, bl=basepar, leg = [0.7, 0.4]**

**COMP\_GEO**

Compare geometry files.

format: **COMP\_GEO, PFF1, PFF2 [, UNIT = unit] [, ERROR = error] [, FULL = full]**

where:

- PFF1*,  
*PFF2*
- File ID for first PFF file
  - File ID for second PFF file
- Note:** Routine assumes first dataset is the bounding box, second is the grid and third to end are the conductors to be compared.
- Note:** If the number to blocks and grid values do not match then the comparison is terminated.

Keyword Parameters:

- UNIT*
- UNIT number if output is to a file
- ERROR*
- This can be a scaler or a vector. If a vector, the first value will be used for grids and the second for conductors.
- GRIDS:
- If set, two grid values will be equal if they differ by less than  $RELEERROR * \text{grid spacing}$  where  $REKERROR = \text{error}/100$  (ie error is taken as a percent)
- CONDUCTORS:
- If set, two conductor values will be equal if they differ by less than  $RELEERROR / (2^{17}) * (\max(x) - \min(x))$  where x represents xi, xj, and xk for each dataset.
- FULL*
- All differences will be printed. ERROR will be used to determine if a difference exists.

Outputs:

Output file differences between two geometry files.

Examples:

Compare files with file id's 1 and 3 and write output to unit.

```
openw, unit, /get, 'difference.dat'
COMP_GEO, 1, 3, unit =unit
free_lun, unit
```

## COMTE

Calculate the standard deviation between two WDFarrays. This is designed for comparison of theoretical and experimental data.

The amplitude of the second waveform is determined by first filtering the second waveform with a low pass filter, ie.,

```
filt, wdf2, out, delta = 0.361*fwhm(wdf2, /quiet, base =0)
```

mx(out) is adjusted to equal mx(wd1) and the standard deviation (SD) is calculated using the x values from the waveform with the fewest number of points (np) in the region of overlap. The SD is calculated:

$$SD = \sqrt{\text{total} (wd1 - wd2)^2 / (np-1)}$$

**Note:** SD is calculated using actual waveform values and not the filtered values. WD1 and WD2 are not modified.

**format:** COMTE, wd1, wd2 [, BASE = base] [, TSHIFT = tshift] [, AMP = amp] [, SDEV = sdev] [, RSDEV = rsdev] [, SCALE = scale] [, QUIET = quiet] [, LEG = leg] [, /NOLEG] [, OUT = out] [, DELTA = delta] [, RDELTA = rdelta] [, CORRELATE = correlate] [, TITLE = title [ , \_EXTRA = extrastuff]

where:

- wd1, wd2,* - WDF array numbers for the arrays to be compared  
WD1 is expected to be theory with low noise.  
WD2 is expected to be experiment. WD2 is moved to out for the calculations. xfr,wd2,out
- BASE* - if present and not zero the following is performed:  
sub, out, 0, base
- TSHIFT* - if present and not zero wd2 is time shifted tsh, out, tshift
- SCALE* - If scale is specified then wd2 will be scaled by this value mul, out, 0 scale  
If scale is not specified or zero then delta or rdelta will be used to filter the data and the resultant amplitude used to scale wd2 to wd1
- DELTA* - if specified wd2 will be filtered with a delta given by delta. ie filt, out, del = delta  
Default = 0.0
- RDELTA* - If delta is zero, then wd2 will be filtered with a delta given by rdelta\*fwhm(wd2, b = 0 , /quiet)  
ie filt, out, del = rdelta\*fwhm(wd2, b=0, /quiet)  
Default = 0.361  
The default value was chosen to give an 0.1% change in amplitude for a gaussian pulse. If RDELTA = 0 and DELTA = 0 or missing, then no filtering will be performed.
- AMP* - returned with the calculated value of amp so that mx(out) = mx(wd1)
- SDEV* - Standard deviation
- RSDEV* - Standard deviation relative to the peak amplitude (mx(wd1))
- LEG* - a two element vector specifying the location for the best fit parameters on the plot as a fraction of the plot range. Default value is [0.04, 0.94]
- NOLEG* - Nonzero value indicates that the no best fit parameters will be printed on the plot, If this

- keyword is zero or omitted, the best fit parameter will be output to the graphics window using the values of leg.
- OUT* - The modified values of wd2 are stored in out  
Default value = get\_maxwdf()
  - QUIET* - If present and not zero, no output will be made
  - CORRELATE* - Correlation coefficient between the two arrays
  - TITLE* - If present this is used for the plot title
  - \_EXTRA* - Any keyword understood by the plo command

**Examples:**

Compare array 1 to array 2 with no baseline adjustment

**COMTE,1,2**

Compare array 3 to array 6 with a filter given by 0.5 of the fwhm.

**COMTE, 3, 6, rdel = 0.5**

**COND\_RANGE**

Determine conductor extent at a plane in space

**format:** COND\_RANGE, A, X, Y, Z [, IER] [, XRANGE = xrange] [,YRANGE = yrange]  
[, ZRANGE = zrange]

where:

- A* - IDL conductor structure or structure array number  
(type = 4)

**Note:** Only 2 of the following parameters will be passed with valid data. The remaining parameter will be returned with the minimum and maximum values of the conductor. If no conductors are found

- X* - First coordinate value or range of values where the conductor extent is desired
- Y* - Second coordinate value or range of values where the conductor extent is desired.
- Z* - Third coordinate value or range of values where the conductor extent is desired

**Note:** An error tolerance of  $2^{-16}$  is used in determining whether a plane crosses a conductor, that is,

$\text{maxp} = \text{max}(\text{a.plane}(1, *, *))$

$\text{minp} = \text{min}(\text{a.plane}(0, *, *))$

$\text{err} = 1.525888\text{e-}5 * (\text{max}([\text{maxp}-\text{minp}, \text{abs}(\text{maxp}), \text{abs}(\text{minp})]))$

- IER* - Error parameter  
0 if maximum gt minimum+err  
1 if manimum eq minimum

-1 if no conductors are found

Keyword parameters:

*X,Y,ZRANGE*

- axis range (standard IDL meaning)

Note: If multiple keywords are set, only the first detected will be used in order: Xrange, Yrange, Zrange

Examples:

Determine the conductor radial extent for conductor structure 4 at theta = 0, and z =4

**COND\_RANGE, 4, r, 0, 4, /x**

Note: r(0) will be the minimum value of the conductor and r(1) will be the maximum value of the conductor.

## CONV

Calculate the convolution of a signal with a response function. For a discussions of the convolutions see "Convolutions" on page 1-5.

Note: If *wd3* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

Note: Caution should be exercised in using this routine. The FFT can produce distortions at the beginning and end of the array.

format: **CONV, wd1,wd2 [, wd3]**

where:

*wd1*

- WDF array of a true signal

*wd2*

- WDF array of a response function

*wd3*

- optional WDF array. If present,

$wd3 = wd1 \otimes wd2$ , else

$wd1 = wd1 \otimes wd2$ , where

$\otimes$  is the convolution operator

Example:

Calculate the expected signal on a detector if the detector response is in array 5 and the calculated signal is in array 6; store the result in array 6

**CONV, 2, 5, 6**

## CREATEPFF

Create a PFF file for output. Current file pointer is set to this file.

format: **CREATEPFF [, filename] [, fileid ]**

where:

*filename*

- name of the file to be created

Default value is 'PFFdefault.pff'

*fileid* - user specified file id  
 (If *fileid* is specified, it will be used as the file id. If *fileid* is not provided or is zero, a value will be provided; default value is the first unused integer  $\geq 1$ . If *fileid* is provided but has been used for another file, the program will inquire if the user wishes to close the old file, to specify a new file id, or to accept the default.

**Examples:**

Create a PFF file, "test" with a default file id

```
CREATEPFF, 'test'
```

Creates a PFF file, "TEST.DAT" with a file id of 66

```
CREATEPFF, 'test.dat', 66
```

Create a PFF file, "PFFdefault.pff" with a default file id

```
CREATEPFF
```

## CUR

A wrapper for IDL cursor which prints the coordinates for the point.

**format:** CUR [, x] [, y] [, \_EXTRA = extrastuff]

where:

*x, y* - Coordinates of the cursor location  
*\_EXTRA* - any valid keywords for the IDL cursor command

**Examples:**

Print the location of a point on a graph

```
CUR
```

## CURP

A wrapper for IDL cursor which prints the coordinates of the point.

**format:** CURP [, P] [, \_EXTRA = extrastuff]

where:

*P* - Vector coordinates of the cursor location  
*\_EXTRA* - any valid keywords for the IDL cursor command

**Example:**

Print the location of a point on a graph

```
CURP
```

## CYL2CART

Convert a cylindrical geometry to a retilinear geometry. The result will be a type 8, finite element mesh in x and y. Three vector data will be converted to rectilinear data. The three vectors are assumed to be Vr, Vtheta, and Vz. They are transformed to Vx, Vy, and Vz. Scaler data (single vector) data is not modified.

**format:** STR2 = CYL2CART (STR1, [,Z VALUE] [, INDEX = index] [, /PSEUDO]  
[, CLABEL = clabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, SAVE = save]  
[, HELPME = helpme])

where:

*STR2* - Field structure with Cartesian data on a finite element mesh. If SAVE is set, STR2 = 0, that is the following executed.

i2s, str1, save, /no\_copy & return, 0

If error occurs, STR2 = -1

**Note:** To check for an error in CYL2CART, use the following.

str2 = cyl2cart(str1, zvalue)

if (size(str2, /type) lt 8) then if str2 eq -1 then print, 'Error occurred in CYL2CART'

**Note:** If a structure is returned or if 0 is returned, then no error.

*STR1* - Structure array number of field structure with Cylindrical data on a rectilinear (r, thets, z) mesh

*ZVALUE* - Slice value in the Z direction.

If STR1 is a two dimensional field then this value is ignored. If STR1 is a three dimensional field then this value indicates the coordinate value for the fixed dimension in the two dimensional plots. Linear interpolation is used according to the value of ZVALUE. If ZVALUE has two elements the average value over the interval used.

**Note:** If Zvalue is not valid, then the closest Z value is used.

**Note:** If INDEX keyword is set, the units are taken as array indices which range from 1 to the maximum value.

**Caution:** Multiple block data has produced unexpected results.

**Note:** If the two elements of ZVALUE span a block boundary, then only the values in the block containing the midpoint will be used. If the midpoint falls on a block boundary, then the results is probably not what you want!!

*INDEX* - If set, units of ZVALUE are Z indices in each block

- which can range from 1 to n\_elements(z)  
 Default : INDEX = 0
- XLABEL** - X Label Value to used X1LAB  
 Default: X1LAB = 'X'
- YLABEL** - Y Label Value to used X2LAB  
 Default: X2LAB = 'Y'
- CLABEL** - Comment value to be used CLABEL  
 Default: CLABEL = 'Cartesian' +str1.clab
- PSEUDO** - If set (/p), a pseudo cylindrical plot is made.  
 This is limited to single block, scalar, 2D data. X is assumed to run from (MIN(X) < 1) to N. Y is time or some similar quantity. The first value will be put on the positive Y axis and the remaining points will be uniformly spaced around the circle. If PSEUDO <= 2 then N = MAX(str1.x1) else N = PSEUDO. If PSEUDO has 2 elements then they will be taken as the minimum and maximum values.
- OFFSET** - If PSEUDO is set, the radius (Y value) will be reduced by OFFSET. This value will be returned in OFFSET.
- SAVE** - If set, save STR2 in a structure array. Valid values are [1, get\_maxstr()] If SAVE is set,  
 then save = save if positive and <= get\_maxstr()  
                   = first unused structure array number  
                   = get\_maxstr() if all arrays are full
- HELPME** - If set, a help message will be printed

**Examples:**

Convert structure f to fxy at the value of Z = 4.5

```
fxy = CYL2CART(f, 4.5)
```

Convert structure 5 to a cartesian structure and store the result in structure 45, 46 and 47 for Z-planes 1, 2 and 3..

```
for i =1, 3 do junk = CYL2CART(5, i, /index, save = 44+i)
```

**Note:** To get vectors in a plane. Vx component in this example and save in 21. To see the plot

```
plotstr, str2, 1, work = 21, cn = 2
```

or to just get the data

```
plotstr, str2, result = 21, /nf, /quiet
```

(results in 128 by 128 array)

(change nfarray to get a different number)

## DECON

Calculate the deconvolution of a signal given the detector response function. For a discussions of the PFIDL convolutions see "Convolutions" on page 1-5.

**Note:** If *wd3* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**Note:** Caution should be exercised in using this routine. The FFT can produce distortions at the beginning and end of the array and the deconvolution process amplifies any high frequency noise in the measured signal.

**format:** DECON, *wd1*, *wd2* [, *wd3*] [, FILTER = *filter*] [, PLOT = *plot*]

where:

- |               |   |
|---------------|---|
| <i>wd1</i>    | - WDF array of a measured signal which has been distorted by a response function  |
| <i>wd2</i>    | - WDF array of a response function  |
| <i>wd3</i>    | - optional WDF array. If present,<br>$wd3 = wd1 \oslash wd2$ , else<br>$wd1 = wd1 \oslash wd2$ , where<br>$\oslash$ is the deconvolution operator |
| <i>FILTER</i> | - optional Wiener filter multiplier <i>F</i>  |
| <i>PLOT</i>   | - if present and nonzero, plots FFT of the response   |

Example:

Calculate the true signal on a detector if the detector response is in array 5 and the measured signal, distorted by the detector, is in array 2; store the result in array 6. Use a default filter to reduce the high frequency noise.

DECON, 2, 5, 6, /filt

## DELSTR

This routine clears structure arrays. If *STR* is zero or undefined; all arrays are cleared. Otherwise, all valid structure arrays specified by *STR* are cleared.

**format:** DELSTR [, *STR*]

Examples:

clear all structure arrays

DELSTR

clear structure array number 15 and 18

DELSTR, [15, 18]

## DELWDF

Eliminate all data from specified WDF arrays; that is set the number of points to zero, and the comment strings to ' '.

**format:** DELWDF [, *WD1*]

where:

*WD1* - WDF array number or an array of numbers  
If missing or zero, then all arrays are cleared.

**Examples:**

Clear WDF array 3 and 5.

**DELWDF, [3, 5]**

Clear all WDF arrays.

**DELWDF (or) DELWDF, 0**

## DIF

Differentiates a WDF array. This procedure uses a simple difference scheme between adjacent points, reduces the number of points by 1, and time shifts the data a half time step later. This is shown in the equation below where  $dx$  is the separation between points.

$$y'(i + 1/2) = (y(i+1) - y(i)) / dx$$

On most data the procedure, SMO or FILT, should be used to reduce high frequency noise.

**Note:** If *wdf2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wdf1*.

**format:** DIF, *wdf1* [, *wdf2*]

where:

*wdf1*, *wdf2* - WDF array numbers  
If *wd2* is present, then  $wd2 = DIF(wd1)$   
else  $wd1 = DIF(wd1)$

**Examples:**

Differentiate array 2 and store the result in array 2

**DIF, 2**

Differentiate array 1 and store the result in array 2. Smooth the result with a 50 ns low pass filter.

**DIF, 1, 2**

**filt, 2, d = 50e-9**

## DIGITIZE

Digitize a curve.

This program must be run on CS6

WEDIT may be used to clean data.

**format:** DIGITIZE, *x*, *y*, *np*, *xover* = *xover*, *yover* = *yover*

where:

*x* - abscissa values

- y* - ordinate values
- np* - number of points
- xover* - xvalues to be used as a guide
- yover* - ordinate values to go with *xover* to be used as a guide.

**Note:** To generate conductors for simulations

1. Outline all conductors at one time.
2. Use TQCOND, ,x ,y, to generate the conductor structures
3. Use TQEDIT (or OSEEDIT), /cond to edit

(Split conductor and Delete Point can be used to separate the conductors)

**Example:**

Calculate the expected signal on a detector if the detector response is in array 5 and the calculated signal is in array 2; store the result in array 6

**CONV, 2, 5, 6**

## DIR

this routine prints a directory of pff files

**format:** DIR [, P1] [, P2] [, P3] [, PAGE = page] [, RESET = reset]  
[, FONT = font] [, XSIZE = xsize]

where:

- p1, p2, p3* - are identical to dirpff
- page* - is the number of lines on the directory; default is directorypage
- reset* - if present and not zero, opens a new directory window. Use this if you wish to leave the current directory window open and put the current directory command in a second window.

**Note:** By using the reset qualifier, a window can be opened for each file with the filename at the top.

- font* - is the font for the directory. Default font is directoryfont which can be set in userqsinit. If reset is specified, font is switched to the current value of directoryfont
- xsize* - is the width of the window in pixels (generally not set)

**Examples:**

Print a directory of PFF datasets 1 to 25 in the current directory with 10 datasets showing

**DIR, 1, 25, page = 10**

Print a directory of the current file

**DIR**

Print a directory of PFF datasets in the file with a file id of 3 located  $\pm 5$  dataset from the current dataset pointer

**DIR, 3, -5**

Print a directory of PFF datasets located  $\pm 5$  datasets from the current dataset pointer in the current directory.

**DIR, -5**

Print a directory of PFF datasets 4 to 34 in the file with a file id of 3. Put the directory in a second window leaving the existing window with its directory showing.

**DIR, 3, 4, 34, /r**

Print a directory of all PFF datasets in the file with a file id of 5 having the letters "mspin" in the first 5 characters of the dataset comment

**DIR, 5, '^mspin'**

Print a directory of PFF datasets having the letters "pin" in the first 64 characters of the dataset comment in the current file

**DIR, 'pin'**

## DIRP

This routine prints a directory of pff files or allows the user to look at one "page" of data at a time.

**format:** DIRP [, INIT] [, FILEID = fileid] [, PAGE= page] [,/HELP] [, LUOUT = luout]

where:

- |               |   |
|---------------|---|
| <i>init</i>   | - initial dataset number<br>default is current dataset                          |
| <i>page</i>   | - number of datasets per page   |
| <i>fileid</i> | - file id of the file   |
| <i>help</i>   | - puts a widget on the screen to guide the user                                 |
| <i>luout</i>  | - writes the directory to a file - luout is the file name and must be a string. |

## DIRPFF

Print a directory listing of datasets in a PFF file. The current file pointer is not changed. The dataset which was last read is also listed.

**format:** DIRPFF [, fileid ] [, low] [, high ]

**format:** DIRPFF [, fileid ] , string

where:

- fileid* - file id of the file for which the directory listing is requested. If *fileid* is zero or omitted, the current directory is printed.
- low, high* - indicates a subset of the directory that is to be printed. If *low* is less than zero, *high* must be omitted and a directory listing of  $\pm$  *low* dataset entries about the current dataset pointer will be listed. If *low* is greater than or equal to zero, then datasets entries between *low* and *high* will be listed. If *high* is zero or negative, *high* is set to the final dataset entry in the file. If *low* and *high* are missing or zero, the entire directory listing of datasets is printed.
- string* - a string variable or a literal string used as a search string. All datasets containing this string (case insensitive) in the first 64 characters of the dataset comment will be printed. If the first character of *string* is a '^', then the string must appear at the start of the comment.

Examples:

Print a directory of PFF datasets 1 to 25 in the current directory

**DIRPFF, 1, 25**

Print a directory of PFF datasets in the file with a file id of 3 located  $\pm 5$  dataset from the current dataset pointer

**DIRPFF, 3, -5**

Print a directory of PFF datasets located  $\pm 5$  datasets from the current dataset pointer in the current directory.

**DIRPFF, -5**

Print a directory of PFF datasets 4 to 34 in the file with a file id of 3

**DIRPFF, 3, 4, 34**

Print a directory of all PFF datasets in the file with a file id of 5 having the letters "mspin" in the first 5 characters of the dataset comment

**DIRPFF, 5, '^mspin'**

Print a directory of PFF datasets having the letters "pin" in the first 64 characters of the dataset comment in the current file

**DIRPFF, 'pin'**

**DIRS**

Show the structures in the qs common area which contain data

format: **DIRS** [, **SMIN**] [, **SMAX**] [, **TRUNCATE = truncate**] [, **XSIZE = xsize**]  
 [, **PAGE = paage**] [, **RESET**]

where:

*SMIN* - first structure array number to be listed or an array of structure array numbers or a QS structure

or

if a single parameter, SMIN can be a search string. Those arrays containing the string in the dataset will be listed. The full set of wild card options is available. See GET\_MATCH

*SMAX* - Last structure array number

*TRUNCATE* - If set, the dataset comments will be truncated at ' - ' This is a toggle; If not specified the last value will be used. Default value = 0 -- no truncate.

*XSIZE* - Width of the window in pixels( generally not set)

*PAGE* - Number of lines on the directory

*RESET* - If set, a new window will be opened.

Outputs:

Structure parameters in a widget.

Examples:

See the contents of all structure arrays

**DIRS**

See the structure with E-field in the comment

**DIRS, 'E-field'**

See the full contents of structure arrays 1 through 13

**DIRS, 1, 13**

or

**DIRS, 1+indgen(13)**

**DIRW**

Show theWDF arrays in the common area which contain data

format: **DIRW** [, **WMIN**] [,

**wMAX**] [, **TRUNCATE = truncate**] [, **FILE**] [, **XSIZE = xsize**]  
 [, **PAGE = paage**] [, **RESET**]

where:

*WMIN* - first waveform array number to be listed or an array of WDF array numbers

or

if a single parameter, WMIN can be a search string. Those arrays containing the string in the dataset will be listed. The full set of wild card options is available. See GET\_MATCH

*WMAX*

- Last WDF array number

*TRUNCATE*

- If set, the dataset comments will be truncated at ‘ ; ‘  
This is a toggle; If not specified the last value will be used. Default value = 1-- truncate comments at first “ ; “.

*FILE*

- If set file will be printed. If file <0 filename will be on a new line. This is a toggle; if not specified the last value will be used. Default value = 0

*XSIZE*

- Width of the window in pixels( generally not set)

*PAGE*

- Number of lines on the directory

*RESET*

- If set, a new window will be opened.

Outputs:

Wveform array parameters in a widget.

Examples:

See the contents of all structure arrays

**DIRW**

See the structure with E-field in the comment

**DIRW, ‘E-field’**

See the full contents of structure arrays 1 through 13

**DIRW, 1, 13**

or

**DIRW, 1+indgen(13)**

See all datasets that begin with a v and the associated files

**DIRW, ‘^v’, / file**

## DIV

Divide a WDF array by another WDF array at corresponding abscissa values. Linear interpolation is used to put the arrays on the same abscissa values. The result only contains values in the region of overlap of the two arrays.

**Note:** If *wdf3* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wdf1*.

**Note:** *TSH* and *NEWEND* can be used to add zero values to the beginning and end of a WDF array.

**Note:** In the case of division by zero, the quotient is set to zero (0.0).

format: **DIV, wdf1, wdf2 [, wdf3] [, CLABEL = clabel] [, XLABEL = xlabel]  
[, YLABEL = ylabel]**

where:

- wdf1, wdf2, wdf3* - WDF array numbers  
If *wd3* is present, then  $wd3 = wd1 \div wd2$   
else  $wd1 = wd1 \div wd2$
- CLABEL* - Dataset comment for the sum  
Default is the dataset comment for *wd1*
- XLABEL* - X-axis label for the sum  
Default is the x-axis label for *wd1*
- YLABEL* - Y-axis label for the sum  
Default is the y-axis label for *wd1*

Examples:

Divide array 1 by array 2 and store the result in array 3

**DIV, 1, 2, 3**

Divide array 3 by array 4 and store the result in array 3

**DIV, 3, 4**

Divide each element of a WDF array by a constant.

**Note:** To divide two constants, use IDL directly. (See last example.)

**format:** **DIV, wd1, 0, value [, wd3]**

where:

- value* - constant or first element of an array
- wd1, wd3* - WDF array numbers  
If *wd3* is present, then  $wd3 = wd1 \div value(0)$   
else  $wd1 = wd1 \div value(0)$

Examples:

Divide all the elements in array 1 by 10 and store the result in array 3

**DIV, 1, 0, 10, 3**

Divide each elements in array 3 by the variable, *shift*, and store the result in array 3

**DIV, 3, 0, shift**

Divide arrays 1, 2, 3 by array 4 and store the results in arrays 5, 6, 7

**for i=1, 3 do div, i, 4, 4+i**

Divide variable, *area*, by variable, *length*, and store in *width*

**width = area / length**

## DIVSTR

Divide a field structure array to another field structure array or by a constant. Store the result in the initial field structure array or another structure array. All attribute or vector quantities are added.

**format:** DIVSTR, *str1*, *str2*, [, *str3*] [, CLABEL = *clabel*]

**format:** DIVSTR, *str1*, 0, *var*, [, *str3*] [, CLABEL = *clabel*]

where:

- str1* - array number of the first field structure array
- str2* - array number of field structure to be divided into first
- str3* - array number for the quotient, if not specified the quotient is stored in STR1 (the first array)
- CLABEL - Optional label for the combined structures.

or

- 0, *var* - Variable or constant to divide into the first structure

**Note:** The following restrictions apply.

*str1* array must be defined and have valid data.

*str2* or 0 and a Variable must be defined.

*str1* and *str2* must both be field arrays and have the same number of vector components.

The SIZE of *str1* and *str2* must be the same.

The values of the spatial coordinates must be equal to about 3e-5; that is,  $\max(\text{abs}(\text{difference}))/\max(\text{value})$  is less than 3.1e-5. The check on spatial coordinates is done globally rather than on a block by block basis.

If a structure has three dimensions, but is degenerate in one of them, then that axis value need not be equal.

**PROCEDURE:**

If STR2 is specified:

Each attribute, or vector component, of STR1 is multiplied to each attribute of STR2

If 0, VAR is specified:

VAR is multiplied to each attribute of STR1

In either case, the product is stored in STR3, if specified, or STR1, if STR3 is not specified.

The quotient is set to zero for all values of *str2* less than 1.525e-5 of the maximum value of each attribute.

**Examples:**

Divide structure array 2 by 3 and store the result in structure 8

**DIVSTR, 2, 3, 8**

Divide the variable OFFSET into structure array 2 and store the quotient in structure 8

**DIVSTR, 2, 0, offset, 8**

Divide structure array 3 by 8 and store the result in structure 10 with a dataset comment, 'Total Impedance - Voltage/current'

**DIVSTR, 3, 8, 10, c= 'Total Impedance - Voltage/current'**

## Droop

This procedure makes a correction for integration droop and magnetic field diffusion. The calculation involves the following calculation.

**Note:** The math is based on the same assumptions as in XDAMP. The implementation is linear in the number of points and is more than 10 times faster than XDAMP at 1000 points and more than 200 times faster at 20000 points.

**format:** **DROOP, wdfin, e\_fold [, wfdout] [, CLABEL = clabel] [, XLABEL = xlabel] [, YLABEL = ylabel]**

where:

- wdfin* - WDF array number of the array to be corrected
- e\_fold* - time constant to use in correcting the data  
**Note:** A negative value will decrease the signal A positive value will increase the signal
- wfdout* - optional output wdf array. If missing or undefined, then the correct array will be stored in wdfin
- CLABEL* - replace the dataset comment for the output array with the string variable specified
- XLABEL* - replace the dataset x-axis label for the output array with the string or string variable specified
- YLABEL* - replace the dataset y-axis label for the output array with the string or string variable specified

Procedure:

All droop corrections are performed on uniformly spaced data. If the data is not uniform it is put on a uniform spacing using linear interpolation. YNEW is the droop corrected value for YOLD. If dt is the point spacing and n is the number of Y values, then

```
const = dt/e_fold
scale = (dt*e_fold)/(e_fold-dt)
exp_fun = exp(-(1+findgen(n-1))*const)
```

and

```
ynew(0) = yold(0)*scale/dt
for i= 1, n-1 do ynew(i) = yold(i)*scale/dt + scale/e_fold*total(exp_fun
(0:i-1)*reverse(ynew(0:i-1)))
```

Examples:

Correct the data in wdf array 1 for 300 ns integrator and store in wdf array 11

**DROOP, 1, 300e-9, 11**

## **ENDPFF**

Close all PFF files and stop the spawned process which handles PFF files. If data is written to a file, either CLOSEPFF or ENDPFF should be used to ensure that file is properly closed with directory information appended to the end of the file.

**format: ENDPFF**

**Examples:**

Close all files and end current PFF session

**ENDPFF**

## **ENDPOST**

Close the postscript output file and set the graphics output to the X windows.

**format: ENDPOST**

**Examples:**

End postscript output and send the next plots output to the terminal

**ENDPOST**

## **ENVELOPE**

This routine calculates the local maximum and minimum of an input WDF array and returns the envelope and the period of the maxima and minima. The envelope uses the maxima and the negative of the minima. The period combines the positive and negative periods.

**Procedure:**

0. Put data on a uniform time base if necessary.
1. Calculate the first derivative.
2. Filter the derivative with a boxcar filter of width given by smooth.
3. Determine the points where the derivative switches sign.
4. Determine the average number of points between zero derivatives = M.
5. Use FITNP or M/8 points to determine a parabolic fit to the data at each zero derivative point.
6. The least square fit results are used to determine maximum/minimum and the center point of the peak.

**Note:** If the parabolic best fit has the wrong curvature or does not converge, no maxima or minima is recorded.

**Outputs:**

The minima are multiplied by -1 and merged with the maxima to provide the result

**Note:** WMAX = MAXWDFARRAY

Maxima are stored in WMAX or MAXVAL

Minima are stored in WMAX-1 or MINVAL

Period is stored in WMAX-2 or PERIOD

Positive period is stored in WMAX-3 or PPERIOD

Negative period is stored in WMAX-4 or NPERIOD

This routine uses the IDL user function, SVDFIT.

**format:** ENVELOPE, wd1 [, wd2] [, SMOOTH = smooth] [, PLOT = plot]  
 [, FITNP = fitnp] [, PERIOD = period] [, PPERIOD = pperiod]  
 [, NPERIOD = nperiod] [, MAXVAL = maxval] [, MINVAL = minval]

where:

- |                 |  |
|-----------------|--|
| <i>wd1, wd2</i> | - WDF array numbers. If wd2 is present, the final curve is stored in wd2 otherwise the result is stored in wd1   |
| <i>SMOOTH</i>   | - If present and not zero, the derivative is smoothed using this width. Default is smooth = 25; minimum for smooth is 11.  |
| <i>plot</i>     | - If present and not zero, the original curve and the extrema are plotted.   |
| <i>FITNP</i>    | - If present, this number of points is used to fit the extrema. Default value is average zero crossing spacing (M) divided by 8. (for a sine wave this will give an error less than 1.0e-4) Unsmoothed data is used for the least squares fit. If FITNP lt 5 then fitnp is returned with the value used. |
| <i>PERIOD</i>   | - WDF array for the period, default = WMAX-2   |
| <i>PPeriod</i>  | - WDF array for the positive period, default = WMAX-3  |
| <i>NPeriod</i>  | - WDF array for the negative period, default = WMAX-4  |
| <i>MAXVAL</i>   | - WDF array for the maximum values, default = WMAX   |
| <i>MINVAL</i>   | - WDF array for the minimum values, default = WMAX-1   |

Examples:

Fit an envelope to the data points in array 2; return the envelope to array 3 and calculate the period. Fit the data between 9 and 100 to an exponential and plot the fit.

```
ENVELOPE, 2, 3 & print, 'Period = ', ave(80, /q)
lim, 3, 9, 100 & wdfit, 3, 4, /ytyp, /plot
```

Fit an envelope to the data in WDF array 4, store the envelope in array 9 and plot the original data and the envelope.

```
ENVELOPE, 4, 9, /plot
```

## EQ\_CHARGE

Function returns the fraction of a given charge species in equilibrium at a given energy.

**format:** `fraction = EQ_CHARGE(z, q, energy [, ier]`

where:

- |                 |   |
|-----------------|---|
| <i>z</i>        | - ion z, ie number of protons must be 3, 5, 6, 7, 8, or 9 and mass asumed to be 7, 11, 12, 14, 16, and 19 |
| <i>q</i>        | - atom net charge, must be greter than or equal to zero and less than or equal to Z                       |
| <i>energy</i>   | - ion kinetic energy or an array of energies  |
| <i>ier</i>      | - optional error flag   |
|                 | 0 no errors   |
|                 | 1 data incomplete or bad  |
|                 | -1 errors present (data out of range)   |
|                 | -99 element not supported   |
| <i>fraction</i> | - charge fraction   |
|                 | -1.0 returned for errors  |

### Examples

find the charge fraction lithium(+1) at 5 Mev and store in frac

```
frac = EQ_CHARGE (3, 1, 5, ier)
```

Plot the charge fraction versus energy for lithium (+2) between zero and 10 MeV

```
x = findgen(201)/20
```

```
plot, x,EQ_CHARGE(3, 2, x, ier)
```

Store the charge fractions for lithium in WDF arrays 10 to 13 and plot them in an overlay plot between 0 and 10 MeV

```
x = findgen(201)/20
```

```
xlab = 'Energy (MeV)' & ylab='Charge Fraction' & tit=ylab +' vs '+xlab
```

```
for i = 0, 3 do i2w,x,EQ_CHARGE( 3, i, x), $
```

```
i+10, xlab=xlab, ylab= ylab, cl= 'Charge - '+strtrim(i, 2)
```

```
plo, 10, 4, /ov, /co, title = tit
```

## FACTORS

Returns the the factors of integers

**format:** `fact = FACTORS(x [, NFACTORS = nf])`

where:

- |                 |  |
|-----------------|--|
| <i>x</i>        | - number for which factors are desired. For this routine: $X = \text{Long}(XIN)$ where XIN is the value given. |
| <i>NFactors</i> | - Number of factors found  |

Example:

Find the factors of 500

```
ff = FACTORS (500, nf = nf)
print, nf
5
print, ff
2      2      5      5      5
```

## FALLT

Calculate the time of a pulse. XY data is converted to a uniform time base (See RiseT for rise time.)

**format:** fall = FALLT (WDF [, STARTVALUE = startvalue] [, ENDVALUE = endvalue]  
[, SMOO = smoo] [, VALUE = value] [, MAXIMUM = maxamp]  
[, MAXTIME = maxtime]

where:

- |                   |  |
|-------------------|--|
| <i>fall</i>       | - Fall time of the pulse between startvalue and endvalue       |
| <i>WDF</i>        | - WDF array for the calculation                                |
| <i>STARTVALUE</i> | - Beginning value as a fraction of the peak,<br>Default = 0.90 |
| <i>ENDVALUE</i>   | - End value as a fraction of the peak.<br>Default = 0.1        |
| <i>SMOO</i>       | - Width for smoothing of the data<br>Default = 5               |
| <i>VALUE</i>      | - Vector with actual times for start and end values            |
| <i>MAXIMUM</i>    | - Maximum value used to determine start and end value          |
| <i>MAXTIME</i>    | - Time at maximum value  |

Example:

Print, the 10% to 90% rise time of wdfarray 5

```
print, riset(5)
```

## FILM\_EXP

Convert film density to relative exposure using a particular film response curve.

**format:** FILM\_EXP, STR1 [, STR2] , RESPONSE = response

where:

- |             |  |
|-------------|--|
| <i>STR1</i> | - Structure array number of film image in density            |
| <i>STR2</i> | - Structure array number of film image in relative exposure. |

*RESPONSE* - Ignored since we only have one response curve.  
(D7)

Example:

Convert structure 1 to relative exposure and store in structure 2

**FILM\_EXP, 1, 2**

## FFTF

Returns the Fourier transform (FFT) of a WDF array. If the WDF array has  $np$  points spaced by increments  $dt$ , then each mode is spaced by  $1/(np*dt)$  and the maximum frequency is  $0.5/dt$ . The FFT is defined by:

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) \exp\left[-j2\pi u \frac{x}{N}\right] .$$

As noted in the definition of the FFT, a sine wave with an amplitude of 1.0 has a maximum amplitude of 0.5 for the positive and negative mode numbers.

**format:** **FFTF, wd1, fftarray [, freq]**

where:

*wd1* - WDF array number  
*fftarray* - complex Fourier transform  
*freq* - frequency information (abscissa values for *fftarray*)

**Note:** The FFTF is designed for returning IDL arrays for further processing. See FFTW for returning the transform to WDF arrays.

**Note:** The IDL FFT command is most efficient if the number of points is  $2^N$ , but for one dimensional arrays, this should not be a major consideration. A 512x512 array requires less than 3 seconds on an HP9000/720; a 513x513 array, more than 11 seconds and a 509x509 array over 2 minutes. The number of points can be changed, using linear interpolation, with the CHA command, or by limiting the domain of the function with the LIM command.

Examples:

Plot the amplitude of the first n modes of the Fourier transform of WDF array 1

**FFTF, 1, array & plot, abs(array), xr=[0,n]**

Plot the amplitude of the FFT of array 2. In this example SPECTRUM is a complex array with the values of the Fourier transform.

**FFTF,2, spectrum, frequency**

**n = n\_elements(frequency)**

**plot, shift(frequency, n/2), shift(abs(spectrum), n/2)**

**Note:** ABS in the last line of this example can be replaced by FLOAT or IMAGINARY to plot the real and imaginary portions of the FFT.

**Note:** This could be done more simply with the following command:

**FFTW, 2, 3 & plo, 3**

## FFTW

Returns the Fourier transform (FFT) of a WDF array to WDF arrays. If the WDF array has  $np$  points spaced by increments  $dt$ , then each mode is spaced by  $1/(np*dt)$  and the maximum frequency is  $0.5/dt$ . The FFT is defined by:

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) \exp\left[-j2\pi u \frac{x}{N}\right] .$$

As noted in the definition of the FFT, a sine wave with an amplitude of 1.0 has a maximum amplitude of 0.5 for the positive and negative mode numbers.

**format:** **FFTW, wd1, amp, phase, real, imaginary, /positive, /degree, /mode**

where:

- wd1* - WDF array number of the array to be transformed.
- amplitude* - Magnitude of the complex transform array
- phase* - Phase angle of the transform = atan(imag, real)
- real* - Real portion of the complex FFT
- imaginary* - Imaginary portion of the complex FFT
- positive* - If present and nonzero, the FFT is restricted to positive frequencies and the amplitude of the nonzero frequency components are multiplied by 2
- degree* - If present and nonzero, the phase angle, *phase*, is returned in degrees.
- mode* - If present and nonzero, the abscissa is the mode number and positive is set to 1.

**Note:** The parameters, *wd1*, *amp*, *phase*, *real*, *imaginary*, must be variables or numbers representing WDF arrays (1 to 82), or zero.

**Note:** The FFT command is most efficient if the number of points is  $2^N$ , but for one dimensional arrays, this should not be a major consideration. A 512x512 array requires less than 3 seconds on an HP9000/720; a 513x513 array, more than 11 seconds and a 509x509 array over 2 minutes. The number of points can be changed, using linear interpolation, with the CHA command, or by limiting the domain of the function with the LIM command.

Examples:

calculate the real and imaginary components of the fft of array 2 and store the result in arrays 10 and 11

**FFTW, 2, 0, 0, 10, 11**

calculate the Fourier modes of array 3 and plot the amplitude of the first 50

modes.

FFTW, 3, 4, /mode & plo, 4, xr = [0, 50]

calculate the amplitude of the fft of array 3 and store the positive frequency components in array 4. Print the peak amplitude.

FFTW, 3, 4, /pos & print, mx(4)

calculate the amplitude and phase of WDF array 1 into arrays 2 and 3 store the phase in degrees; find the maximum positive frequency.

FFTW, 1, 2, 3, /deg & a = mx(2,L=0,/quiet,time=freq) & print, freq

## FILT

Provides band pass, high pass and low pass Fourier filters of WDF arrays. See section 1.3.1 for more details.

(FILT calls LOP, HIP, BAP, or BAR for actual filtering)

**Note:** If *wd2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**Note:** Caution should be exercised in using this routine.

**format:** **FILT**, *wd1* [, *wd2*] [, **MODE** = *mode*] [, **DELTA** = *delta*] [, **FREQUENCY** = *freq*] [, **WIDTH** = *width*] [, **BAND**= *band*] [, **LOW** = *low*] [, **HIGH** = *high*] [, **ORDER** = *order*] [, **TYPE** = *type*] [, **ATTEN** = *atten*] [, /BP] [, /BR] [, /HP] [, /PLOT]

where:

*wd1*, *wd2* - WDF array numbers  
If *wd2* is present, then  $wd2 = \text{FILT} ( wd1 )$   
else  $wd1 = \text{FILT}( wd1 )$

*MODE*, *DELTA*, *FREQUENCY*

- specify the cut off frequency of the high and low pass frequency filters and can be used to specify the center of the band pass and band reject filters. Only one of these parameters is required since they are related according to the relation:  $mode = (np*dt)^{\dagger} / delta = frequency * (np*dt)^{\dagger}$  where *delta* is the wavelength or period, and *frequency* is the frequency. If none of the parameters are provided, then *mode* is set to 12.

*WIDTH*, *BAND*

- specify the width of "band-type" filters in terms of modes (*width*) or frequency (*band*) and are related by the relation:  $width = band * (np*dt)^{\dagger}$ . If no width is specified for a "band-type" filter, then *width* is set to  $0.8*mode$ .

*LOW*, *HIGH*

- specify the width of "band-type" filters in terms of frequencies. These values are converted to modes

using the relations:  $mode = (high + low) * (np*dt)^{\dagger}$   
and  $width = (high - low) * (np*dt)^{\dagger}$ .

- ORDER** - specifies the order of the filter; default is 2.
- TYPE** - Indicates the type of filter desired; valid values are 'BUT', 'EXP', 'IDE' and 'USE' indicating Butterworth, exponential, ideal and user supplied filter types. Only the first three characters of the string are checked and 'USE' is the default if no match is found on the first three strings; the default if TYPE is omitted is 'BUT'.
- ATTEN** - filter function which must be provided with a TYPE of USER. For the other filter types, the calculated filter function is returned in *atten*. *atten* must have the same number of points as the WDF array *wd1*.
- PLOT** - nonzero value indicates the filter function is to be plotted vs. frequency.
- HP** - nonzero value indicates a high pass filter
- BP** - nonzero value indicates a band pass filter
- BR** - nonzero value indicates a band reject filter

$^{\dagger}np$  is the number of points,  $dt$  is the point spacing

**Note:** For Fourier filters assume a periodic function. If the initial and final points of the array are not at the same value, large distortions of the dataset will occur at the extremes of the dataset.

**Note:** For Butterworth filters, a *delta* equal to the full width at half maximum of a gaussian pulse will only attenuate the pulse several (~4) percent.

**Note:** To look at dominant modes in a WDF array n, use the commands:

**FFTF, n, y & plot, abs(y)**

To look at the first 50 modes:

**plot, abs(y), xr=[0,50]**

Examples:

Filter array 1 with a Butterworth low pass filter and a delta of 15 ns (freq = 6.667e7) and store the result in array 3

**FILT, 1, 3, d = 15e-9 (or filt, 1, 3, freq = 6.66667e7)**

Filter array 1 with an exponential low pass filter with a 3db frequency of 100 MHz and store the result in array 3

**FILT,1, 3, freq = 1e8, type = 'exp'**

Filter array 1 with a band pass filter between 10 and 100 MHz and store the result in array 3

**FILT, 1, 3, /bp, low=1e7, high =1e8**

Filter array 1 with a band reject filter between 80 and 100 MHz and store the result in array 3. Use a filter order of 4, plot the filter function and return it in farray.

```
FILT, 1, 3, /br, low=8e7, high =1e8, order = 4, /plot , atten = farray
```

## FULFNAM

Return the full file name from input string FNameIn. This routine handles the following file specifications:

format: **FULFNAM, FNameIn, Full\_Name**

where:

*FNameIn* - Input name  
*Full\_Name* - Output name

This routine handles the following file specifications:

$\$ddd$ /file,  $\$(ddd)$ /file - Where “ddd” is an environment variable pointing to a directory  
 $\{ddd\}$ /file  
 $\$fff$ ,  $\$(fff)$ ,  $\{fff\}$  - Where “fff” is an environment variable pointing to a file  
 $\$(xxx)yyy$ ,  $\{xxx\}yyy$  - Where “xxx” is an environment variable and entire string points to a file  
 $\sim$ /file - For file specification relative to user’s HOME directory

## FWHM

Function returns the full width at half maximum (fwhm) of a WDF array. The maximum is the maximum deflection from the baseline.

format: **width = FWHM(wd1 [, BASE = base] [, LEFT = left] [, RIGHT = right] [, QUIET = quiet] [, PEAK = peak])**

where:

*width* - full width at half maximum; if no value is found, zero is returned.  
*wd1* - WDF array number  
*BASE* - indicates a baseline value; if omitted, the baseline is indicated by the user with the cursor or set to zero if quiet is nonzero.  
*LEFT* - indicates the left edge of the region for the analysis; default value is the first data value  
*RIGHT* - indicates the right edge of the region for the analysis; default value is the last data value

*QUIET* - if present and nonzero, indicates no outputs is to be sent to the terminal. If zero, the waveform will be plotted with the baseline and the half-maximum amplitude and the values of these parameters.

#### Examples:

Print the fwhm of array 2 assuming a baseline value of zero

```
print, FWHM (2, b = 0)
```

Calculate the fwhm of array 2 between abscissa values [2, 4] with an assumed baseline offset of 0.852; "wid" is an IDL variable.

```
wid = FWHM(2, base = 0.852, R= 4.0, L = 2)
```

## GET\_ARRAY

this function returns an array from a structure

format: `GET_ARRAY(structure, space [, block] [, lab= lab] )`

where:

*structure* - IDL structure name or structure array number type must be

- 1, field
- 2, particle
- 3, grid

*space* - if not zero indicates spatial dimension desired, if negative indicates field data and attribute data.

*block* - indicates block - not used for particle data

*lab* - attribute label or block label associated with this data

#### Example:

make a contour plot of field data in structure, fld1

```
contour, GET_ARRAY(fld1,-1), get_array(fld1, 1, l=xl), $
GET_ARRAY(fld1, 2, l= yl), xtit= xl, ytit= yl
```

**Note:** if fld1 is a 3 dimensional array then the above becomes

k = xx (define k to be the appropriate location)

a = get\_array(fld1,-1)

```
contour , a(*, k, *) , GET_ARRAY(fld1, 1, la=xlab),$
GET_ARRAY(fld1, 3,la=ylob), xtit = xlab, ytit = ylab
```

## GET\_DIR

This routine returns up to three arrays of dataset numbers, dataset types, dataset comments and the entire directory line. The size of these arrays is returned in number.

format: `Format: GET_DIR [,p1] [, p2] [, p3] [, DATASET=dataset] [, TYPE=type] [, COMMENT=comment] [, NUMBER= number] [, POINT= point] [, LINE = line] [, FILENAME = filename]`

where:

*p1, p2, p3* - have the same possibilities as *DIRPFF*

That is:

[, *fileid*] [, *low*] [, *high*]

or [, *fileid*], string

where:

*fileid* - file id of the file

*low* - first dataset number

If *low* lt 0 then make a directory +/- *low* around current dataset

*high* - last dataset number

*string* - search string for the dataset comments

*DATASET* - integer array of dataset numbers

*TYPE* - string array of dataset type

*COMMENT* - string array of dataset comments

*LINE* - string array of the entire dataset directory line

*NUMBER* - number of elements in the above arrays

*POINT* - dataset pointer

*FILENAME* - name of the pff file

Examples:

Get a directory around current datasets or 5 datasets

`GET_DIR[, -5] [, line = line]`

Get a directory of datasets 95 to 100

`GET_DIR[, 95] [, 100] [, line = line]`

Get a listings of all datasets with 'MOCAM2?'

`GET_DIR[, 'MOCAM2?'] [, line = line]`

## GET\_DSET

Get PFF dataset numbers for all dataset comments containing a specified string. - 1 indicates no dataset found.

**Note:** Dataset comments in the directory are currently limited to 64 characters. A search string longer than this will result in no matches.

**format:** `maxwdf = Map= GET_DSET(string [, FILEID = fileid] [, MAXDSET = maxdset]`

where:

*string* - search string

if string = ' ' or ' ' or '\*' then

Map = 1 + lindgen(ndspff(fid)) (MAXDSET is ignored)

The following special characters can be used:

\* - Wild card indicating 0 to n characters

To use a \* in a search string use “\\*”

? - Wild card indicating 1 and only 1 characters

To use a ? in a search string use “\?”

^ - If this is the first character, indicates that the string begins the dataset comment. All other times a ^ is taken literally. To begin a search string with a literal ^ use: “\*^” or “\^” All search strings are assumed to begin with “\*” unless a “^” is present.

\$ - If this is the last character, indicates that the string ends the non-blank portion of the dataset comment. All other times a \$ is taken literally. To end a search string with a literal \$ use: “\*\$” or “\\$” All search strings are assumed to end with “\*” unless a “\$” is present.

*FILEID*

- file id

Default is 0 (the current pff file)

*quiet*

- if set, no messages will be printed

*MAXDSET*

- maximum number of files desired. If more than maxdset files are found, then an additional search is performed over the returned datasets with the search string modified to force an exact match; ie. if not present a “^” will be added to front of the string and a “\$” to the end of the string.

Example:

Find all the datasets with VDIODE in the comment.

```
wdf = GET_DSET('vdiode')
```

Find all the datasets with no-blank characters only VDIODE.

```
wdf = GET_DSET('^vdiode$')
```

Find all the datasets with the string VDIODE and at least one more character at the end

```
wdf = GET_DSET('vdiode?')
```

Read all the datasets in the current file starting at position 1

```
re, 1, GET_DSET()
```

or more simply

```
re, 1, ‘ ‘
```

## GET\_FCUP

Propagate an ion pulse to a target assuming that they are moving with a velocity corresponding to a specified voltage pulse. The energy versus arrival time is calculated for protons by default. The charge collector signals are then transported back to the anode and the currents modified to reflect the rising or falling voltages using the ratio of the arrival times. If the slope of the arrival time changes slope it is set to zero. The transit time is smoothed using a 5 point boxcar by default. Present version limits the response to the high energy portion.

**Note:** The voltage pulse is propagated forward. The measured current is then corrected for the transit time of projected back in time.

**Note:** This is based on `get_vprop`. Velocity is assumed to be given by:

$$\text{velocity} = \sqrt{2.0 * 1.6022e-19 / 1.6726e-27 * Vc * Q / M}$$

where  $M = 1, 12, 16$  protons, carbons and oxygen, respectively,  $Q$  is the charge, and  $Vc$  is the acceleration voltage

**format:** GET\_FCUP, Vcor, Fcup, OUTPUT, [, Nsecond = nsec] [, DISTANCE = distance] [, IONMASS = ionmass] [, CHARGE = charge] [, OUTPUT = output] [, DELTAT = deltat] [, SMOOTH = smooth] [, LIMIT = limit] [, /HighVolt] [, /LowVolt]

where:

- |  |  |
|--|--|
| <i>Vcor</i>  | - wdf array for the corrected voltage or a search string XY data is sorted - Process is terminated if multiple voltages at the same time.  |
| <i>Fcup</i>  | - wdf array(s) for the charge collectors or a string for the search  |
| <i>OUTPUT</i>  | - If present and a valid wdf number, this will be used in place of the keyword parameter. If output is a single number and fcup is several then successive arrays will be used. Output may be an array if fcup is an array. This parameter of the keyword must be provided |
| <i>Nsecond</i>   | - abscissa units for vcor<br>Default is nsecond = 1, abscissa in nanoseconds   |
| <i>DISTANCE</i>  | - ion propagation distance in meters<br>Default distance = -0.3 m  |
| <i>IONMASS</i>   | - Ion mass in amu. Default = 1<br>IONMASS > 1  |
| <i>CHARGE</i>  | - Ion charge - Default = 1; Charge > 1   |
| <b>Note:</b> Only one ion mass and one charge can be used. |  |
| <i>SMOOTH</i>  | - Smoothing for the transit time.<br>Default = 5   |
| <i>OUTPUT</i>  | - output WDF arrays. Ignored if positional parameter is present  |

<i>LIMIT</i>	Voltage cut; eliminate voltages below this value. Default is 10% of peak $0 < \text{limit} < 0.9$
<i>HighVolt</i>	- If set only high voltage portion of the arrival time will be saved. This is the default.
<i>LowVolt</i>	- NOT CURRENTLY ALLOWED. If set only the low voltage portion of the arrival time will be saved. This is ignored if HighVolt is set. Default is to return the high energy arrival time curve.

**Examples:**

Calculate the current of protons measured at a charge collector 30 cm away. Assume the energy is in wdf array 3, the fcup in 4, the store the result in 10

```
GET_FCUP, 3, 4, out = 10
```

```
GET_FCUP, 3, 4, ion =1, char =1, /high, out = 10, dist = 0.30
```

Calculate the current of Li<sup>+</sup> ions measured at charge collectors 15 cm away using 'VLIDJJ' as the corrected voltage and put the results beginning in array 40. The charge collectors are in 'Fcup\_1c', ;fcup\_2c'

```
GET_FCUP, 'vlidjj', 'fcup_*c', dist = .15, out =40, ion =7
```

Calculate the current of protons at the anode from a diagnostic 40 cm away. Assume the energy is in wdf array 3 and store the results beginning in 40. Use all wdf arrays with 'fcup' in the comment. Return only the high voltage branch of the arrival time curve. Limit results to voltages over 25% of the peak.

```
GET_FCUP, 3, 'fcup', dist = .40, /high, ion =1, limit = 0.25, out = 40
```

**GET\_MATCH**

From a string array return the array elements which contain a specified string. This may be used for WDVALID, STRVALID, and GET\_WDF.

```
format: array_numbers = GET_MATCH (string, STRING_ARRAY
[, NELEMENTS = nelements] [, WDFARRAY = wdfarray]
[, STRUCTURE = structure] [, CASESEN = casesen])
```

where:

<i>array_number</i>	- an array of integers representing the array elements which contain STRING If no matches are found then -1 is returned. If keywords wdfarray or structure then the array numbers are returned.
<i>string</i>	- search string, if it is an array only the first element is used.

string = ' ', '\$', '^', '^\$', '\*' will return all possible elements

**Note:** String may end in blanks.

for example: 'cat ' is different form 'cat' the latter would allow not only 'cat dog' but also 'cat1', 'cat2'...

All trailing and leading blanks are stripped from the string\_array. Therefore 'cat ' would not find 'cat ' in string\_array; the correct string would be: 'cat\$'

The following special characters can be used:

- \* - Wild card indicating 0 to n characters. To use a \* in a search string use "\\*"
- ? - Wild card indicating 1 and only 1 character. To use a ? in a search string use "\?" **Note:** '^?'
- ^ - If this is the first character, indicates that the string begins the dataset comment. All other times a ^ is taken literally. To begin a search string with a literal ^ use: "\*\*^" or "\^". All search strings are assumed to begin with "\*" unless a "^" is present.
- \$ - If this is the last character, indicates that the string ends the non-blank portion of the dataset comment. All other times a \$ is taken literally.

**Note:** The above words "non-blank". A search string of the form ' cat \$' will simply be 'cat\$'. To end a search string with a literal \$ use: "\\$". All search strings are assumed to end with "\*" unless a "\$" is present.

*STRING\_ARRAY* - String array. Each element of string array is searched for STRING. If not present, then WDFarray or STRUCTURE must be set.

**Note:** string\_array = strtrim(string\_array, 2) but the input value is not modified.

*Nelements* - Number of array elements for which a match is found.

*WDFARRAY* - If set, WDF dataset comments will be used for STRING\_ARRAY. If WDFARRAY has two elements, they will be taken as the bounds for the

search.

- STRUCTURE** - If set, structure array dataset comments will be used for STRING\_ARRAY. If STRUCTURE has two elements, they will be taken as bounds for the search.
- CASESEN** - If present and not zero, then search is case sensitive. Default is CASESEN = 0 (not case sensitive)

Example:

List the wdf arrays beginning with M and ending in 8

```
print, GET_MATCH('^m*8$', /wdf)
```

List the wdf arrays beginning with M and ending in 8 and containing a total of 7 characters

```
print, GET_MATCH('^m?????8$', /wdf)
```

Print the elements of titles which contain the letters msp and end in cc

```
print, titles( GET_MATCH('msp*cc$', titles))
```

Technique:

1. All input strings with '\*' are converted to '?'
2. Search string is moved to a work string.
3. LENWORK is used as a pointer array to the first character of each substring.
4. LENSTR is a length indicator for the length of each substring. EXCEPT LENSTR is negative if a ? is found. The negative value of LENSTR is the number of successive ? and thus the number of characters to skip.
5. If a \* follows a ? then more characters can be skipped. Lenwork is set negative if an \* follows a ?, otherwise it is positive
6. Leading \* are ignored so '\*^' is the same as '\^'
7. Blanks in string\_array are only meaningful if they have non-blank characters on both sides.

## GET\_MAXSTR

This function returns the maximum number of structure arrays

format: number = GET\_MAXSTR]

**Note:** The last two arrays are used by some procedure as work arrays

## GET\_MAXWDF

Function returns the maximum number of a WDF arrays.

format: maxwdf = GET\_MAXWDF ()

where:

*maxwdf* - maximum number of WDF arrays

Example:

Determine the maximum number of arrays possible with this version of PFIDL

```
print, GET_MAXWDF()
```

## GET\_RESP

Calculate the response function of a detector given the measured and actual signals. For a discussions of the convolutions see "Convolutions" on page 1-5.

**Note:** If *wd3* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**Note:** Caution should be exercised in using this routine.

format: GET\_RESP, wd1,wd2 [, wd3] [, FILTER = filter] [, PLOT = plot]

where:

*wd1* - WDF array of a measured signal which has been distorted by a response function

*wd2* - WDF array of a undistorted, true signal

*wd3* - optional WDF array. If present,  
 $wd3 = wd1 \oslash wd2$ , else  
 $wd1 = wd3 \oslash wd2$ , where  
 $\oslash$  is the deconvolution operator

*filter* - optional Wiener filter multiplier F

*plot* - if present and nonzero, plots FFT of the true signal

Example:

Calculate the response of a detector if the measured signal from the detector is in array 5 and the true, undistorted signal is in array 2; store the result in array 6. Use a default filter to reduce the high frequency noise.

```
GET_RESP, 2, 5, 6, /filt
```

## GET\_STR\_VALUE

Get field values at points

**Warning:** This routine is a preliminary version but should work!!

**Warning:** The maximum structure array will be used as a working structure array and will contain the two dimensional data used for the contour plot unless the keyword WORK provides a valid working structure number.

**Note:** Xarray, yarray, and Farray are the most used values for this routine. Put desired x and y values into Xarray and yarray and get the result in farray.

format: GET\_STR\_VALUE, A [,FLD] [, KVALUE1] [, VALUE1] [, FARRAY = farray]  
 [, NFARRAY = nfarray] [, XARRAY = xarray] [, YARRAY = yarray]  
 [, TITLE =title] [, XTITLE = xtitle] [, YTITLE = ytitle] [, BLOCK = block]  
 [, ASPECT = aspect] [,MULTIPLY = multiply] [, ADD = add]

[, KINTEGRATE = kintegrate] [, HEADER = header]  
 [, SETDEFAULT = setdefault] [, UNSET =unset]  
 [, SHOWDEFAULT = showdefault] [, INDEX = index] [, SMOOTH = smooth]  
 [, TRANSPLOT = transplot] [, QUIET = quiet] [, LEFT = left] [, RIGHT = right]  
 [, RESULT = result] [, WORK = work] [, NO\_COPY = no\_copy]  
 [, XRANGE = xrange] [, RANGE = yrange] [, CNTOUR = cntour]  
 [, XINTEG = xinteg] [, YINTEG = yinteg] [, START = start] [, REVERSE = reverse]  
 [, XLINE = xline] [, YLINE = yline] [, NPOINTS = npoints] [, WIDTH = width]  
 [,PLINE = pline] [, OUT = out] [, XYDATA = xydata] [, CLINE = cline]  
 [, CZERO = czero] [, CABSCISSA = cabscissa] [, CMAXIMUM = cmaximum]  
 (any stuff) (any contour keywords -- see IDL manual-- examples include:  
 FILL, X(Y)TYPE, POSITION, X(Y)MARGIN, THICK, FOLLOW, DOWNHILL,  
 X(Y)TICKLEN, X(Y)STYLE, C\_ANNOTATION, C\_CHARSIZE, C\_LINestyle, ....)  
 NOTE: FILL and DOWNHILL are new feature of IDL

where:

- A* - IDL field structure (a.type = 1) or QS array number
- FLD* - Field or attribute parameter to be plotted.  
**Note:** If FLD is an array, the square root of the sum of the squares will be used.  
**Note:** If A has only one attribute (such as an image or a charge density); then this parameter can be omitted.
- KVALUE1* - If FLD is a two dimensional field then this value is ignored. If FLD is a three dimensional field then this value indicates the coordinates which are fixed for the two dimensional plots. For 3D plots, x and y are chosen according to:  
 $x = (kvalue1 \bmod 3) + 1$  and  $y = (kvalue1 + 1) \bmod 3 + 1$
- VALUE1* - If FLD is a two dimensional field then this value is ignored. If FLD is a three dimensional field then this value indicates the coordinate value for the fixed dimension in the two dimensional plots. Linear interpolation is used according to the value of VALUE1. If VALUE1 has two elements the average value over the interval is used.  
**Note:** If INDEX keyword is set, the units are taken as array indices which range from 1 to the maximum value.  
**Note:** If two elements of VALUE1 span a block boundary, then only the values in the block containing the midpoint will be used. If the midpoint falls on a block boundary, then the result is probably not what you want!!!

Keyword Parameters:

- BLOCK* - If the field is multi-block, BLOCK can be used to specify one or more blocks to be plotted/integrated/...
- ASPECT* - If set, will attempt to scale axis based on xrange and yrange
- MULTIPLY* - Multiply the field by a constant or a vector
- ADD* - Add a scaler to the field before plotting  
**Note:** If F is the field then (multiply\*F + add) will be plotted
- KINTEGRATE* - If present and not zero, integrate the slice rather than average over the extent of value1(0) to value1(1)  
 The integral will be performed between value1(0) and value1(1)  
 If integrate = 1, a simple integral  
 If integrate = 2, a x dx integral  
 If integrate = 3, a x<sup>2</sup> dx integral  
 Limitations:  
     Data must have a single block  
     Two values of value 1 must exist
- HEADER* - If specified a plot header, consisting of the current data and the current file name, is displayed in the lower left corner of the plot. See the HEADER command to set default  
 HEADER parameters (character size, spacing, alignment)  
 Default is HEADER = 1  
**Note:** For X-windows screen dumps, many will want to increase the character size with the command:  
 HEADER, /set, charsize = 0.8  
 (Default is 0.6 of current size)
- SETDEFAULT* - If present and not zero, then the following parameters will be saved and used as the default values in all future calls to PLO. If a keyword is specified, it will always be used rather than the saved values.
- UNSET* - If present and not zero, will set all saved values of keywords to their default values.  
 Normal default values are listed below.
- | Keyword | Default value           |
|---------|-------------------------|
| HEADER  | 1                       |
| COLOR   | FLDCOLOR (-1 typically) |
| XRANGE  | 0                       |

	YRANGE	0
	NLEVEL	20
	LEVELS	0
	INDEX	0
	THERMOMETER	0
	ASPECT	0
<i>SHOWDEFAULT</i>	- Show the present default plot values.	
<i>INDEX</i>	- If present and not zero, then the units of VALUE1 will be taken as indices of the field array. Valid values are 1 to the maximum number of values in KVALUE1. Care should be exercised with multiple block data	
<i>SMOOTH</i>	- If present and not zero, the two dimensional field will be smoothed using a boxcar filter. The number of cells will be the lowest odd number greater than or equal to the maximum of [3, smooth]	
<i>FARRAY, NFARRAY</i>	- For single block data FARRAY is the field quantity being plotted and NFARRAY is ignored. For multiple block data, a uniform grid is generated with NFARRAY by NFARRAY elements. The field quantity is interpolated to this grid. <b>Note:</b> All plots, except contour plots, plot the array, FARRAY. Default value of NFARRAY is 128 if NFARRAY is undefined, of $\text{abs}(\text{NFARRAY}) > 16$ if NFARRAY is specified. (That is “/nf” would result in a value NFARRAY = 16)	
<i>TITLE, XTITLE, YTITLE</i>	- If these are character strings, they will be used as the plot title and the axis labels. If they are not character strings or are undefined (i.e., delvar, title, xtitle, ytitle) then they will be returned with the values actually used for the plots. The default TITLE is the attribute label and the dataset comment. The default axis labels are the PFF axis labels read with the data.	
<i>TRANSPLOT</i>	- If present and not zero, interchange ordinates and abscissa. SETDEFAULT can be used to set a default value	
<i>QUIET</i>	- If present and not zero, level information will not be printed. If quiet has a value of 1 (/QUIET or QUIET =1), NO plots will be made. If QUIET is not zero and not 1, then plots will be made, but not output to the screen. Default: QUIET = 0	
<i>LEFT</i>	- Indicates a new plot using the left axis. A single axis	

will be drawn on the left side of the grid. For this plot XMARGIN will be set to XMARGIN = [!X.margin(0), !X.margin(0)] and YSTYLE = 8.

**Note:** If LEFT is set, OVERPLOT is set to 0.

**Note:** Enter a value of ystyle of 8 plus the desired value if any other ystyle values are desired.

Example:

Make a series of lineouts and overplot them on the contour plot. Average the second spatial component between [0,1 ] and take lineouts at .13, .14, .15.

```
plotstr, 1, 1, 2, [0,1], xl = [0.13,.14,.15], /pl, /left
plo, 1, 3, /over, /right, /ns, /color, thick = 3
```

### *RIGHT*

- Indicates a OVERPLOT making and using the right axis. If RIGHT is present and not zero, then: A single axis will be drawn on the right side of the existing grid. The field data will be overplotted on the existing grid. Before using this command, a plot should be created using the keyword: /LEFT

**Note:** If RIGHT is set then OVERPLOT and IGNORE are set to 1.

**Note:** After this command all memory of the original y-axis scaling is lost

### *WORK*

- Structure array number to be used as a working array.  
Default is maxstructure.

**Note:** If NoSave is set then work will not be saved.

### *NO\_COPY*

- If this keyword is set then the plotted data will not be saved in WORK. Use for very large arrays.

### *CNTOUR*

- If present and not zero, a contour plot will be drawn. If no other plot types are selected, a contour plot will be drawn by default.

**Note:** Keyword is cntour or /cn to avoid problems with color, conductor keywords.

### *SAVE*

### *OVERPLOT*

- Save the 3d transform for shadesurf or surface
- If present and not zero, then the existing grid is to be used. This option is ignored if any type of surface plots are made. With this option set, previous values of KVALUE1 and VALUE1 are used by default as well as the choice of abscissa and ordinate coordinates. This option assumes that the previous plot was from a PLOTFLD, PLOTGRD, PLOTCON, or PLOTPAR command and attempts

- use the same parameters for the current plot. To use OVERPLOT with an arbitrary grid, set the IGNORE keyword.
- IGNORE* - This keyword is ignored unless OVERPLOT is set. If OVERPLOT is set and IGNORE is set, then all previous plot information is ignored.
- NLEVEL* - Number of contour plot levels. Default is 20, maximum is 90.
- LEVELS* - Array of values for the plot contours. The actual contour levels will be returned in LEVELS if it is a scalar and NLEVEL is not equal to 1.  
**Note:** If PLOTSTR is used, the levels will NOT be returned because the EXTRA keyword is used to pass this value in a structure.
- C\_LABELS* - Array of integers of value 0 or 1 to indicate if contour lines are to be labeled. Default is no labels on the contour lines. If "/FOLLOW" is used as a keyword C\_label is set to the array [1,0,1,0,1, ...] and every other line is labeled. If C\_LABEL= make\_array(30, /fix, val= 1), then every contour is labeled.
- COLOR* - Array of colors for the contour lines. If color is not an array, then If color = -1 then colors will go from 9 to !d.table\_size uniformly spaced using the existing color table If color = -2 then the 7 primary colors will be used to make a color map. With 20 levels we will have 3 red, 3 green, 3 blue, 3 purple, 3 orange, 3 light blue, and 2 yellow. If color = -3 the 7 standard colors will be cycled. If color is positive, that color will be used for all contours, [1, 7] are the primary colors, 9 to !d.table\_size depend on the color map. Color = 0 is the default plot color ie white or black.  
 Default is set by SETDEFAULT or QSINIT  
 The colors are: red(1), green(2), blue(3), purple(4), orange(5), light blue(6), yellow(7).
- XRANGE, YRANGE* - Two element arrays indicating the desired axis scaling. If they are scalar or undefined, the actual data range will be returned in these variables.
- CONDUCTOR* - Specifies a conductor structure or structure array number specifying a conductor structure. If valid, the conductors will be overplotted. Colors and thicknesses can be set with SETDEFAULT.
- GRID* - Specifies a grid structure or a structure array

- number. If valid, the grid will be overplotted. Color and thickness can be set with SETDEFAULT.
- THERMOMETER*
- If keyword is set a thermometer will be generated with the contour plot.  
Default IDL margins are [10, 3] for x and [4, 2] for y  
If TPARAM does not have a Tag Name beginning with 'CU' then: If no tag names begin with 'H' xmargin will be set to [!x.margin(0), 10] otherwise ymargin will be set to [8, !y.margin(1)]  
**Note:** xmargin or ymargin passed to plotfld will override these settings.  
**Note:** If THERMOMETER is a structure, then TPARAM is ignored and set equal to THERMOMETER
- TPARAM*
- Structure containing data for the thermometer, Tag names must be valid for the THERMOMETER procedure.
- XINTEG*
- Integrate the field variable with respect to the abscissa value (x) according to the following values:  
If XINTEG = 0, do nothing.  
If XINTEG = 1, integrate  $f(x, y) dx$   
If XINTEG = 2, integrate  $f(x, y) x dx$   
If XINTEG = 3, integrate  $f(x, y) x^2 dx$   
dx is the spacing between abscissa values. The initial value of the integral is set to 0.0 or to the value in the WDF array specified by START.  
**Note:** If the resultant plot has contour lines passing through conductors, TQCLEAN can be used to improve the appearance of the plot.  
**Note:** If two blocks terminate on a third, then the initial value of the integral at the block boundary is linearly interpolated between the two incoming blocks.
- YINTEG*
- Integrate the field variable with respect to ordinate value (y) according to the following values:  
If YINTEG = 0 or if XINTEG = [1,2,3], do nothing.  
If YINTEG = 1, integrate  $f(x, y) dy$   
If YINTEG = 2, integrate  $f(x, y) y dy$   
If YINTEG = 3, integrate  $f(x, y) y^2 dy$   
dy is the spacing between ordinate values. The initial value of the integral is set to 0.0 or to the value in the WDF array specified by START.  
**Note:** If XINTEG ne 0 then YINTEG is ignored.

**Note:** If two blocks terminate on a third, then the initial value of the integral at the block boundary is linearly interpolated between the two incoming blocks.

*START-*

- If present this is interpreted as initial values at the lower limit of integration. Start is an integer indicating a WDF array. Values are between 1 and maxwdfarray. Any blocks with coordinates in the domain of START will be initialized unless they are connected directly to another block. Values at one point on a boundary are propagated along the entire boundary and linearly interpolated between adjacent values.

*REVERSE*

- If present and not zero, the limits of integration are reversed

*LINEOUT keywords*

- Values of the LINEOUT coordinates are determined in the following sequence.) For lineouts parallel to an axis

1. Two elements in width - These values taken as the max/min value

2. NPOINTS >0, NPOINTS equally spaced lineouts in specified direction(s), spacing is between the specified range -- use xrange and yrange to limit the values of the lineouts.

3. XLINE, YLINE is an array of values for the lineout

**Note:** To take a single lineout at one location width must be specified with two values (which may be equal) and /xline or /yline specified to indicate the direction of the lineout.

or NPOINTS = 1 then one lineout will be taken at the appropriate y or x location specified by XLINE or YLINE with a width specified by WIDTH

**Note:** If the width of a lineout spans a block boundary, only values in the block containing the midpoint of the lineout will be used.

*XLINE*

- If present and not zero, a WDF array of field value vs the abscissa is generated at designated ordinate values.

*YLINE*

- If present and not zero, a WDF array of field value vs the ordinate is generated at designated abscissa values.

*NPOINTS*

- Indicates number of lineouts in the X and/or Y

direction.

**Note:** If  $npoints = 1$ ,  $xline$  and/or  $yline$  are assumed to equal the desired value of the lineout.

- WIDTH* - If a single value, then it is taken as the width of the lineout. Default value is 0. If two values are present, they are taken as the minimum and maximum values for obtaining the lineout. In all cases, the average value between the limits or in the Width about the designated point is used for the lineout.
- PLINE* - If present and not zero, plot a dashed line at the position of the lineouts
- OUT* - Value of the first WDF array for storing lineouts or an array of values for the lineouts. Default is  $OUT=1$
- XYDATA* - If present and not zero, the data will be stored in XY format.  
DEFAULT is  $XYDATA = 1$  (See W2I in the PFIDL help.) Lineouts along conductors
- CLINE* - If present and not zero, a lineout along a conductor will be taken. The value of  $cline$  corresponds to the conductor number in QSEdit. If  $CLINE$  is negative lineouts will be taken of all conductors. The WDF array generated will be stored in WDF array number, out, or the next WDF array following the X and Y lineouts. All waveforms are stored as XY data.
- Note:** To obtain conductor geometries the TQ suite must be used. TQEDIT/QSEdit can be used with the other routines to generate an arbitrary line. To use the actual geometry the following can be used.  
Read geometry - assume conductors in datasets 3, 4, 5, 6  
READSTR, 1, 3, 4  
Put the geometry into the TQ suite in Z-R, Z-X orientation and cut at  $y$  or  $\theta = 0$ :  
TQREADCON, 1, 3, 1, 0  
Edit the geometry if necessary:  
QSEdit, /cond  
Take the lineout along all conductors  
PLOTSTR, 2, 1, 2, 0,  $cline = -1$ ,  $con = 2$   
Technique: If  $dx$  and  $dy$  are less than  $2^{(-14)}$  to a field point then that point is used. If not then the four closest field values are used to calculate a

- weighted average. Values less than  $2^{(-14)} \times \text{range}$  are not used unless CZERO is set. Weight is  $(dx \times dy)^{(-1)}$
- CABSCISSA* - Specifies the independent variable for the lineout. 1 - X-value, 2 - Y-value, 3 - Length= $\sqrt{x^2+y^2}$   
Default = 1
- CMAXIMUM* - Maximum point spacing. Generally only nodes in QSEEDIT will be used but additional points will be inserted if the spacing exceeds cmaximum.
- CZERO* - In general all values less than  $6.10352e-5 \times (\text{range of field})$  are ignored. If CZERO is set then all values will be used.

**OUTPUTS:**

Designated datasets are plotted and WDF arrays generated for lineouts.

**Example:**

Contour plot of the charge density, in structure F, from a 2-D calculation.

**plotfld, f**

**GET\_TIME**

Function returns the abscissa values of a WDF array at a given value of the ordinate using linear interpolation. Zero is returned if no points are found.

**format:** `time = GET_TIME (wdf, yvalue [, NPOINTS = npoints])`

where:

- wdf* - WDF array number
- yvalue* - ordinate value at which abscissa is desired. Zero is returned if yvalue is outside the range of the wdf array
- NPOINTS* - number of abscissa values found. This parameter is optional.

**Example:**

Determine the times when an amplitude is equal to 0.6 in waveform array 3.

**time = GET\_TIME (3, 0.6, np= np)**

**GET\_VALUE**

Function returns the ordinate values of a WDF array at a given value of the abscissa using linear interpolation.

**format:** `yvalue = GET_VALUE (wda, xvalue)`

where:

- yvalue* - IDL variable to be filled with the ordinate value
- wda* - WDF array number

*xvalue* - abscissa value at which ordinate is desired

Example:

Determine the amplitude of array 4 at the 36 times equal to *event(i)*. Store the values in the IDL variable *power(i)* and covert to WDF array after converting to nanoseconds and terawatts.

```
for i = 0, 35 do power(i) = GET_VALUE (4, event(i)) * 1.0e-12
I2W, event * 1.0e9, power, c = 'Power vs. Time', x = 'Time (ns)', y= 'Power (TW)'
```

## GET\_VPROP

Propagate an ion pulse to a target assuming that they are moving with a velocity corresponding to a specified voltage pulse. The energy versus arrival time is calculated for protons, carbon and oxygen by default.

velocity = sqrt( $2.0 \times 1.6022e-19 / 1.6726e-27 \times Vc \times Q / M$ )

where M = 1, 12, 16 protons, carbons and oxygen, respectively, Q is the charge, and Vc is the acceleration voltage

**format:** GET\_VPROP, Vcor, [, Nsecond = nsec] [, DISTANCE = distance]  
[, IONMASS = ionmass] [, CHARGE = charge] [,OUTPUT = output]  
[, DELTAT = deltat] [, LIMIT = limit] [, /HighVolt] [, /LowVolt]

where:

*Vcor* - wdf array for Vcor limited to peak  
*Nsecond* - abscissa units for vcor  
 Default is nsecond = 1, abscissa in nanoseconds  
*DISTANCE* - ion propagation distance in meters  
 Default distance = -0.3 m  
*IONMASS* - Ion mass in amu. Default = 1  
 IOMASS > 1  
*CHARGE* - Ion charge - Default = 1; Charge > 1

**Note:** IONMASS and charge must have the same number of elements or a single element. For example:

```
ionmass = [1, 4, 12, 16], charge = 2      is ok (single Charge)
ionmass = 16, charge = [1, 2, 3, 4]      is ok (single Mass)
ionmass = [1,4,12 12], charge = [1,1,1,2] is ok (same Mass & Charge)
ionmass = [12, 16], charge = [1, 2, 3, 4] is ok (not same or single)
```

*OUTPUT* - output WDF arrays. default are vcor+1, vcor+2...  
*LIMIT* Voltage cut; eliminate voltages below this value.  
 Default is 10% of peak  
 $0 < \text{limit} < 0.9$

*HighVolt* - If set only high voltage portion of the arrival time will be saved. Data saved as uniform data with spacing deltat.

- LowVolt* - If set only the low voltage portion of the arrival time will be saved. This is ignored if HighVolt is set. Data saved as uniform data with spacing deltat. Default is to return the entire arrival time curve.
- DELTAT* - Minimum Time spacing. Used only if HighVolt or LowVolt is set. Default is average initial spacing\*0.2. If Deltat eq 0, then data will be saved exactly.

#### Examples:

Calculate the arrival time of protons at a target 30 cm away. Assume the energy is in wdf array 3 and store the result in 10. Return all possible voltages.

```
GET_VPROP, 3, ion =1, out = 10
```

Calculate the arrival time of protons at a target 40 cm away. Assume the energy is in wdf array 3 and store the result in 4. Return only the high voltage branch of the arrival time curve. Limit results to voltages over 25% of the peak

```
GET_VPROP, 3, dist =.40, /high, ion =1, limit = 0.25
```

## GET\_WDF

Return the wdf number(s) of all wdf arrays having a certain string in the comment. Similar to wdvalid.

**format:** array\_numbers = GET\_WDF(string)

where:

- array\_number* - an array of integers representing the WDF arrays with STRING in the wdf comment. If no matches are found then -1 is returned.
- string* - search string

Keywords:

- CASESEN* - If present and not zero, then search is case sensitive. Default is CASESEN = 0 (not case sensitive)

#### Examples:

List all wdfarrays with data

```
print,GET_WDF()
```

List the wdf arrays having 'mocam' in the dataset comment.

```
print, GET_WDF('mocam')
```

Plot in an overlay all wdf arrays with 'MOCAM2' in dataset comment.

```
plo, GET_WDF('MOCAM2'), /over
```

## GETF

Calculate the frequencies corresponding to the modes of the Fourier transform of a WDF array.

format: **GETF**, *freq*, *wd1*

where:

- freq* - IDL array to hold the frequency values
- wd1* - WDF array number

Example:

Calculate the frequency for each mode of the fft for the data in WDF array 4.

**GETF**, *freq*, 4

## GETX

Return the abscissa values of a WDF array.

**NOTE:** See also WDF2IDL or W2I.

format: **GETX** , *xarray*, *wd1* [, **MAXIMUM** = *maximum*] [, **MINIMUM** = *minimum*]

where:

- xarray* - IDL variable to be filled with the abscissa values
- wd1* - WDF array number
- maximum* - maximum X value
- minimum* - minimum X value

Example:

Store the abscissa values of array 4 in the IDL variable *dist*

**GETX**, *dist*, 4

## GETY

Return the ordinate values of a WDF array.

**NOTE:** See also WDF2IDL or W2I.

format: **GETY** , *yarray*, *wd1* [, **MAXIMUM** = *maximum*] [, **MINIMUM** = *minimum*]

where:

- yarray* - IDL variable to be filled with the ordinate values
- wd1* - WDF array number
- maximum* - maximum X value
- minimum* - minimum X value

Example:

Store the values of array 4 in the IDL variable *amplitude*

**GETY**, *amplitude*, 4

## GETYF

Function returns the ordinate values of a WDF array.

**NOTE:** See also WDF2IDL or W2I.

**format:** `yarray = GETYF (wd1 [, MAXIMUM = maximum] [, MINIMUM = minimum] )`

where:

- yarray* - IDL variable to be filled with the ordinate values
- wd1* - WDF array number
- maximum* - maximum X value
- minimum* - minimum X value

**Example:**

Store the values of array 4 in the IDL variable *amplitude*

```
amplitude = GETYF (4)
```

## GTITLE

Write a general title for a plot at the top of the page. User must set the *y\_margin* before making a plot which is to have a general title.

**format:** `GTITLE [, MyTitle] [, /Multi] [, YMARGIN = ymargin] [, OFFSET = offset] [,CHARSIZE = charsize] [, /SETDEFAULT]`

where:

- MyTitle* - Title for this plot. This is required if default is not set.
- Multi* - If set, the general title is centered on the page. If not set, the general title is centered over the current plot window.
- YMARGIN* - Space above the plot window.  
Default value for GTITLE is [4,4]. IDL default is [4.000, 2.000] which does not leave room for a general title. On the first call to GTITLE the !y.margin will be set to [4,4] unless a value is passed.  
**Note:** This keyword is checked on the first call to GTITLE and later ignored unless the keyword (SETDEFAULT) is set.
- OFFSET* - Offset from the top of the screen. Default is 0.1 characters. Minimum value is .01
- CHARSIZE* - Character size. Default is 1.5. Minimum value is 0.1. If maximum(!p.multi(1:2)) > 2 then the size is scaled by 0.5
- SETDEFAULT* - If set then default values of Title, Charsize, and Offset can be set. The first time GTITLE is called

then setdefault is assumed to be set.

Procedure:

XYOUTS is used to print a general title.

Example:

**Note:** Before making a plot which you wish a general title enter:

**GTITLE;** sets the ymargin to [4,4] the first time called IF the margin is set at [4,2]

**!y.margin = [4,4]**

**GTITLE, /set, ymargin = [4,4];** other parameters may also be set

Put the title 'X-Ray Analysis' on a plot and make it the default title

**GTITLE, 'X-Ray Analysis', /set**

Print the default title in the center of the plot window

**GTITLE**

Print the default title in the center of the page

**GTITLE, /mul**

Print the title 'Special Title' with character size = 2

**GTITLE, charsize =2, 'Specail Title'**

## HAK

HAK is a procedure that performs "Hit any key to continue". It waits for keyboard input, clears the type-ahead buffer and allows the application to continue. If the device is either Postscript or Encapsulated Postscript, the procedure returns with no operation.

**format:** HAK [, MESSAGE = message]

where:

*message* - If present and a string, then print the string.  
 If present and not a string, then print:  
 "Hit any key to continue..."  
 If *message* is not present, print nothing.

Examples:

Plot WDF arrays 1 to 4; pause after each plot and provide the message: "Hit any key to continue..."

**for i = 1, 4 do begin & plo, i & hak, /m & end**

**Note:** This could be accomplished with the single command:

**plo, 1, 4**

## HEADER

Print a descriptive header (actually a footer) at the bottom of a graphics window.

**format:** HEADER [, TEXT] [, CHARSIZE = charsize] [, SPACE = space] [, ALIGN = align] [START = start] [, UNSET = unset] [, SHOWDEFAULT = showdefault] [, SETDEFAULT = setdefault] ;

where:

- text* - optional string array. Typically this will be the date and file name. DEFAULT = 'Date: ' + systime(0)  
**Note:** If SETDEFAULT, SHOWDEFAULT or UNSET is specified then DEFAULT value of TEXT is ''
- charsize* - Character size for XYOUTS. DEFAULT = 0.6  
**Note:** If there are more than two plots in either direction on the page, the character size is reduced by 0.6
- space* - Spacing between lines of text.  
Default = 1.5\*Character Height
- align* - Indicates the alignment of the header.  
0 - DEFAULT - Left side of plot window  
1 - Right side of plot window  
0.5 - Centered in the plot window
- start* - Vertical height of bottom line of text.  
Default = 0.25\*Character Height
- unset* - If present and not zero, will set all the save values of keywords to their default values:  
CHARSIZE = 0.6 (\* Default Value)  
SPACE = 1.5 (\* Character Height)  
ALIGN = 0.0  
START = 0.25 (\* Character Height)
- showdefault* - Show the present default header keyword values.
- setdefault* - If present and not zero, then the following parameters will be saved and used as the default values in all future calls to HEADER. If a keyword is specified, it will always be used rather than the saved values.

Example:

Put the date and file name at the bottom of the current plot

```
STUFF = [ 'Date: ' + systime(0), 'File: ' + filename]
HEADER, stuff
```

Put your name, position, and center on the right side of the plot with full size characters.

```
HEADER, align = 1, ['Tom', 'Chief of Everything', 'Happy Science Center'],
chars=1
```

## HIP

Apply a high pass filter to a WDF array.

**Note:** If *wdf2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wdf1*.

**Note:** See *FILT* for keyword definitions. This routine is called by *FILT*.

**format:** HIP, *wdf1* [, *wdf2*] [, TYPE = *type*] [, ATTEN = *filter*] [, ORDER = *order*] [, MODE = *bin*] [, /PLOT].

where:

*wdf1*, *wdf2* - WDF array numbers  
If *wd2* is present, then *wd2* = HIP(*wd1*)  
else *wd1* = HIP( *wd1*)

Default keyword values are:

*type* = 'BUT'  
*order* = 2  
*bin* = 12

Examples:

Apply a high pass filter to WDF array 1 and store the filtered array in array 2 using the default parameters

HIP, 1, 2

## IDL Functions

Perform standard IDL functions on an array or structure; these functions include absolute value, arc-cosine, natural and base 10 logarithm, arc-sine, arc-tangent with 1 or 2 arguments, hyperbolic sine, cosine and tangent, exponential function, square root, tangent, real and imaginary parts of a complex array, forward and inverse Fourier transform, maximum, minimum, total, complex conjugate, statistical functions (linear regression, analysis of variance, cluster analysis,...), and others.

**Note:** *GETY*, *GETYF*, and *PUTY* are described elsewhere.

Example:

Calculate an array of areas from an array of radii in WDF array 5 and store in array 6.

*xfr*, 5, 6 ; store the abscissa values into array 6  
*area* = !pi \* *getyf*(5)^2 ; note that !pi is an IDL constant for pi  
*puty*, *area*, 6

Use IDL internal functions to operate on structure data

*im* = *get\_array*(1, -3)  
*im* = *sin*(*im*)  
*put\_array*, 1, *im*, -3

Calculate the sine of WDF array 5 times pi/2 and store in array 6.

```
xfr, 5, 6 ; store the abscissa values into array 6
puty, sin (!pi/2 * getyf(6) ), 6
```

Calculate and print the average of WDF array 6.

```
gety, array, 6
average = total(array)/n_elements(array)
print, 'Average of array 6 is ', average
```

## I2S

This function puts a structure into the qs structure common.

**format:** i2s, structure, i

where:

*structure* - structure to be stored in a structure array  
*i* - the structure array number

Example:

Store structure ff into the structure array at location 2

```
i2s, ff, 2
```

## I2W

Converts an IDL array to a WDF array.

**Note:** If *wd1* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then the command is aborted.

**format:** I2W, xarray, yarray, wd1 [, CLABEL = clabel] [, XLABEL = xlabel]  
 [, YLABEL = ylabel] [, FILE = file] [, EPSILON = epsilon] [, DELTA = delta]  
 [, XYDATA = xydata] [, LOWRES = lowres]

where:

*xarray, yarray* - IDL arrays to be converted to a WDF array.  
*wd1* - WDF array number  
*clabel* - dataset comment (used for title of plots)  
*xlabel* - x-axis (ordinate) or block label  
*ylabel* - y-axis (abscissa) label  
*file* - file or source of these dataset values  
*epsilon* - presumed accuracy of the values. If two abscissa values are closer than epsilon in relative accuracy, ie.  $(x_j - x_i) / \max(\text{abs}(x))$  is less than epsilon, then  $x_i$  and  $x_j$  are assumed equal.  
 DEFAULT: EPSILON =  $\min(1.5e-5, 0.5/np)$  where  $np$  is the number of points.  
*delta* - spacing of the uniformly spaced array  
**Note:** If the data is sparse then a small value of

- delta can be specified to insure that all features are resolved. XYDATA may also be used.
- xydata* - if set, indicates the x and y arrays are to be stored exactly as passed. If they are used for add, subtract, etc, then they are converted to a uniform timebase.
- lowres* - if set, reduce the assumed resolution of the input data by a factor of 10. This is for ascii data with only 6 digits input.

**Note:** *xarray* must have at least as many points as *yarray*. If *xarray* is not a monotonically increasing function, the values will be sorted and multiple *yarray* values at a given abscissa will be averaged. *xarray* and *yarray* **will be modified**. If the points in *xarray* are not uniformly spaced, then the data is linearly interpolated using an interval equal to the minimum point spacing in *xarray* or producing an array of 30,000 points, whichever is larger.

Example:

Convert IDL arrays x and y to WDF array 2 after scaling the x values by 1.0e9 and store the title and x-axis labels with the WDF array.

```
I2W, x*1e9, y, 2, c='Voltage - Shot 1234', x='Time (ns)'
```

Convert IDL arrays x and y to WDF array 3 and store the title and axis labels with the array and associate the data with the file, "qsnew.dat"

```
I2W, x, y, 3, c='Pinch Profile', y='Amplitude', x='Position(cm)', f='qsnew.dat'
```

Generate a sine wave of duration 100  $\mu$ s and frequency equal to 50 kHz in WDF array 5. (!pi is single precision pi.)

```
I2w, findgen(1000)*0.1e-6, sin(!pi/100*findgen(1000)), 5
```

## IMAGE

Generate an image file from data which has a reflection plane or has periodic boundaries. The data must be two dimensional or three dimensional with a degenerate dimension and have a single attribute, i.e. it must be a scalar field. (present version only supports FIELD and CONDUCTOR structures)

```
format: IMAGE, STR1 [, STR2] [, REFLECTION = reflection]
        [, PERIODIC = PERIODIC] [, VALUE = value] [, /INVERT]
        [, NPERIODS = NPERIODS] [, CLABEL = clabel] [, /REPEAT]
```

where:

- STR1* - Structure array or array number of the first field structure
- STR2* - Array number or structure array for the result. If not specified the image is stored in STR1 (the first array)

**Note:** If str1 is an array number then str2 will be an array number, and if str1 is a structure array, then

- str2 will be a structure array.
- REFLECTION** - If present must equal to +/- 1, 2, or 3.  
 + indicates maximum value for reflection  
 - indicates minimum value for the reflection.  
 1, 2, 3 indicates coordinates 1, 2, or 3  
 Transform:  $x_{new} = 2 * value - x_{old}$
- PERIODIC** - If present must equal to +/- 1, 2, or 3.  
 Presently adds periods at the maximum value.  
 1, 2, 3 indicates coordinates 1, 2, or 3  
 Transform:  $x_{new} = x_{old} + value$
- VALUE** - Reflection mode:  
 If a value other than the maximum or minimum of the data is required for the reflection, value is used to specify this point. Value must be greater than the maximum spatial value of the data or less than the minimum to avoid multiple values at a point.  
 (Error of  $6.1e-5$  is allowed in this check)  
 Default is the maximum or minimum data value.  
 Periodic mode  
 This is the period of the periodic data. This should be larger than the range of the data.  
 (Error of  $2 * (6.1e-5)$  is allowed in this check)  
 Default value is the maximum data value - minimum data value.  
**Note:** For particle data, value should be defined.
- INVERT** - If present and not zero, multiple the field values by -1.
- NPERIODS** - Number of replications for the periodic data, default = 1
- AGAIN** - If present and not zero, all parameters will be the same as the last call to IMAGE  
**Note:** This means all other parameters are ignored
- CLABEL** - Optional label for the combined structures.

**Examples:**

Assume a field structure in 2 and conductors in 1; generate a reflection about the positive y value in both.

**plotstr, 2, 1, work = 3 ; select the r component of the field**

**IMAGE, 3, reflect = 2 ; reflect the field**

**IMAGE, 1, /again ; reflect the conductors**

Reflect structure array 2 about the maximum y value and store in array 3

**IMAGE, 2, 3, reflect = 2**

Reflect structure array 2 about the minimum Z value and store in array 4,

and invert the field

**IMAGE, 2, 4, reflect = -3**

Reflect structure array 5 about the  $r=0$  value and store the result in array 4

**IMAGE, 5, 4, reflect = 1, value = 0**

Replicate structure array 3 about the maximum theta value (dimension 2) and store the result in array

**IMAGE, 3, period = 2**

Replicate structure array f about the maximum r value and store the result in structure, g

**IMAGE, f, g, period=1**

## INT

Integrates a WDF array. This procedure uses a simple summation scheme, increases the number of points by 1, sets the first point to zero and shifts the initial point a half time step earlier. This is shown in the equation below where  $dx$  is the separation between points,  $n$  is an optional moment parameter, and the primes refer to the original coordinates.

$$y(i + 1/2) = y(i - 1/2) + y'(i) * dx \{n = 0\}$$

$$y(i + 1/2) = y(i - 1/2) + y'(i) * x'(i)^n * dx \{n \neq 0\}$$

**format: INT, wd1 [, wd2] [, MOMENT = moment]**

where:

- wd1, wd2* - WDF array numbers  
If *wd2* is present, then  $wd2 = INT(wd1)$   
else  $wd1 = INT(wd1)$
- moment* - Optional moment for the integral. Default = 0

**Note:** If *wdf2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wdf1*.

Examples:

Integrate array 2 and store the result in array 7

**INT, 2, 7**

Integrate array 1 and store the result in array *energy*. *Energy* will be an integer value between 1 and 42

**INT, 1, energy**

## INTSTR

Integrate a two dimensional structure with respect to its first or second dimension. Three dimensional data will be reduced to a two dimensional structure.

**Note:** Four dimensional data is checked for valid data as closely as 3D data.

**format:** Structure2 = INTSTR (Structure1, Field, Kvalue1, Value1, Kvalue2, Value2)

where:

*Structure1* - Input this is a multidimensional field type structure or a structure array number

*Structure2* - Output this is a two dimensional field structure or a multidimensional field structure with degenerate dimensions.

*Field* - Field component to be sliced

**Note:** Field may be positive or negative; Field = -(abs(Field))

**Note:** If Field is an array, the square root of the sum of the squares will be used.

**Note:** If structure 1 has only one attribute (such as an image or a charge density); then this parameter can be omitted.

*KVALUE1,2* - If Field is a two dimensional field then these values are ignored. If Field is a three or four dimensional field then these values indicate the coordinates which are fixed for the two dimensional plots.

*VALUE1, 2* - If Field is a two dimensional field then these values are ignored. If Field is a three or four dimensional field then these values indicate the coordinate values for the fixed dimensions in the two dimensional plots. Linear interpolation is used according to the value of VALUE1. If VALUE1 has two elements the average value over the interval is used.

**WARNING:** If INDEX is specified, VALUE1 will be changed to spatial coordinates using the midpoints of the bins.

*overplot* - If overplot is not zero and kvalue1,2 is undefined, the values will be taken from the last plot.

*block* - If the field is multi-block, BLOCK can be used to specify one or more blocks to be sliced. BLOCK is an array of one or more values.

*index* - If present and not zero, then the units of VALUEi will be taken as indices of the field array. Valid values are 1 to the maximum number of values in KVALUEi.

Care should be exercised with multiple block data. **WARNING:** If INDEX is specified, VALUE1 will be changed to spatial coordinates using the midpoints of the bins.

*xinteg* - Integrate the field variable with respect to the abscissa value (x) according to the following

values:

If XINTEG = 0, do nothing.

If XINTEG = 1, integrate  $f(x, y) dx$

If XINTEG = 2, integrate  $f(x, y) x dx$

If XINTEG = 3, integrate  $f(x, y) x^2 dx$

$dx$  is the spacing between abscissa values. The initial value of the integral is set to 0.0 or to the value in the WDF array specified by START.

Default: If yinteg and xinteg are both 0 or undefined, then xinteg = 1.

**Note:** If two blocks terminate on a third, then the initial value of the integral at the block boundary is linearly interpolated between the two incoming blocks.

*yinteg* - Integrate the field variable with respect to ordinate value ( $y$ ) according to the following values:  
 If YINTEG = 0 or if XINTEG = [1,2,3], do nothing.  
 If YINTEG = 1, integrate  $f(x, y) dy$   
 If YINTEG = 2, integrate  $f(x, y) y dy$   
 If YINTEG = 3, integrate  $f(x, y) y^2 dy$   
 $dy$  is the spacing between ordinate values.  
 The initial value of the integral is set to 0.0 or to the value in the WDF array specified by START.

**Note:** If two blocks terminate on a third, then the initial value of of the integral at the block boundary is linearly interpolated between the two incoming blocks.

*start* - If present this is interpreted as initial values at the lower limit of integration. Start is an integer indicating a WDF array. Values are between 1 and maxwdfarray

*smooth* - If present and not zero, the two dimensional field will be smoothed using a boxcar filter. The number of cells will be the lowest odd number greater than or equal to the maximum of [3, SMOOTH]

#### EXAMPLE:

Reduce structure  $f$  to a two dimensional structure by slicing along the second dimension between 0.0 and 0.4 and integrate with respect to  $x$ . Save the magnitude of the integrated field vector in structure  $g$ .

```
g = INTSTR(f, [1,2,3], 2, [0.0, 0.4], /xint)
```

Assume  $d$  is a charge density structure with one attribute. Take a slice at  $z=0.25$  and store the integral ( $y dy$ ) in structure array 12.

```
i2s, INTSTR(d, 3, .25, yint = 2), 12
```

or

```
i2s, INTSTR(d, 1, 3, .25, yint = 2), 12
```

Integrate structure array 2, a two dimensional structure, with respect to the first dimension and store the result in  $st2$ .

`st2 = INTSTR(2, /x)`

or

`st2 = INTSTR(2)`

## INVERTCT

Reverse the color table

**Note:** Existing plots are modified, but loading a new color table will change everything back to the original colors. See `swapct. pro` to modify, only the foreground and background colors.

**format:** `INVERTCT [, /RESET] [, /XYDATA] [, /GIFDATA]`

where:

- RESET* - If set, reset the invertct flag
- GIFDATA* - If set, maximum and minimum color table values will be switched and the colors between 9 and `!p.color-1` will be inverted.
- XYDATA* - If set, will only invert colors 9 and `!p.color-1` if both `xydata` and `gif` are set then the colors between 9 and `!p.color-1` will not be inverted.

**Note:** `invertct`, `/xy`, `/gif` is the same as: `swapct`

Technique:

switches the color table entries between the minimum and the maximum (`!p.color`, `!p.background`) < (`!d.table_size-1`). Minimum is zero unless `xydata` is set.

## IPROP

Procedure propagates current to a target and separates the target current into a high energy and low energy branch. The total current at the target is also calculated

**Warning:** The WDF arrays: `MAXWDFARRAY` and `MAXWDFARRAY-1`, are used as scratch areas to calculate the total current. (81 and 82 for current version (10/2/95)) The Arrival Time vs Time and Actual scale vs Time will be stored in these two arrays before leaving.

1. First I and V are put on a common uniform timebase
2. Voltage is smoothed and limited to 8% of peak
3. `Vdiode` is translated to the target.

$$k = 2.0 * 1.602177-19 / 1.66054e-27$$

$$\text{velocity of particle} = \sqrt{k / \text{mass} * \text{vdiode}}$$

$$\text{for a proton mass} = 1.0073$$

$$\text{carbon mass} = 12.0000$$

4. Arrival time derivatives are calculated and limited to two branches.
5. Absolute value of time derivatives is calculated and smoothed with a three

point boxcar.

6. Current is translated to the target.

7. Total current is calculated.

**format:** **IPROP, vdiode** [, **idiode**] [, **vth**] [, **ith**] [, **vtl**] [, **itl**] [, **itotal**] [, **MASS = mass**] [, **DISTANCE = distance**] [, **SMOOTH = smooth**]

where:

<i>vdiode</i>	- WDF array with diode voltage (volts vs nanoseconds)
<i>idiode</i>	- WDF array with diode current (amps vs nanoseconds) Default = vdiode + 1
<i>vth</i>	- WDF array with high energy target voltage. Default = idiode + 1
<i>ith</i>	- WDF array with high energy target current. Default is vth + 1
<i>vtl</i>	- WDF array with low energy target voltage. Deafault is ith + 1
<i>itl</i>	- WDF array with low energy target current. Default is vtl + 1
<i>itotal</i>	- WDF array with total target current. Default is itl + 1
<i>MASS</i>	- Atomic Mass in AMU, Deafault = 1.007276 = proton
<i>DISTANCE</i>	- Distance to target, Default = 0.3 meters
<i>SMOOTH</i>	- Number of points used to smooth the voltage, Default = 5. Set to 0 to get no smoothing.

## JOINW

Join or concatenate two WDF arrays. A value may be specified at which the concatenation occurs with ordinate values for the first specified waveform applying before this point and the ordinate values of the other waveform applying after this point. If the first array has less than 3 data values between the beginning and value then only the second array will be returned. If the second array has less than 3 data values between value and the end of the array then only the first array will be returned. If neither array has 3 data values then an error will occur. If no value is specified, the abscissa values for the two arrays are combined and sorted and duplicate abscissa values are eliminated by averaging ordinate values. The result is linearly interpolated to a uniform array, using the minimum spacing for the two arrays.

**format:** **JOINW, wd1, wd2** [, **wd3**] [, **VALUE = value**]

where:

<i>wd1, wd2</i>	- WDF array numbers of the two arrays to be joined
<i>wd3</i>	- WDF array for the composite

If *wd3* is not present, the composite array is stored in *wd1*.

*value*

- If specified, the first array will be terminated at an abscissa value of *value* and the second array will be ignored before abscissa values of *value*.

Example:

Join WDF arrays 2 and 3 and store the result in array 8

**JOINW, 2, 3, 8**

Join WDF arrays 3 and 5 into one array. Assume array 3 is to be used up to an abscissa value of 1 and array 5 afterwards. Store the result in 3.

**JOINW, 3, 5, value = 1.0**

## JOINSTR

Join a field structure array to another field structure array. Store the result in the initial field structure array or another structure array. All attribute or vector quantities are joined. No check is made to verify blocks are consistent.

**format: JOINSTR, STR1, STR2 [, STR3**

where:

- STR1* - array number of the first field structure array
- STR2* - array number of the field structure to be joined to the first structure
- STR3* - array number for the combination, if not specified the combination is stored in *STR1* (the first array)

Procedure: Blocks of *STR1* are combined with the blocks of *STR2*. The combination of blocks is stored in *STR3*, if specified, or *STR1* if *STR3* is not specified.

Example:

Join structure arrays 2 and 3 and store the result in structure 8

**JOINSTR, 2, 3, 8**

Join structure arrays 3 and 8 and store the result in structure 10 with a dataset comment, 'Total Charge - Line + Diode'

**JOINSTR, 3, 8, 10, c = 'Total Charge - Line + Diode'**

## KILL\_PART

Return the location of killed particles of filter killed particles.

**format: kill = KILL\_PART(str [, FILTER = filter] [, ALLORBITS = allorbits])**

where:

- kill* - a structure of the parameters for killed particles

- str* - particle structure or structure array number
- FILTER* - If present, killed particles will be filtered according to the value of filter and the entire trajectories returned in kill. The form of filter is similar to w1 in plotpar. FILTER must have 3 elements ie. filter = [A, B, C] where A is the quantity. A positive indicates spatial dimensions, and A negative indicates attribute quantities.
- ALLORBITS* - If all the particles in str are known to be killed particles and the user only wants the endpoints, then all orbits will select the endpoints of all orbits.  
**Note:** Filter is ignored if allorbits is set.

Example:

Plot the location of killed particles from the particle structure, str in a z-r plot

```
plotpar, KILL_PART(str), 3, 1
```

Plot the trajectories of all particles which die in the first dimension range [0, .1] in a z-r plot

```
s = KILL_PART(str, filt = [1, 0, .1])
```

```
plotpar, s, 3, 1, /follow
```

Plot the endpoints of the above subset of the particles in r/z

```
plotpar, KILL_PART(s, /all), 3, 1
```

## LAB

Sets the dataset comment of a WDF array.

**format:** LAB, wd1, string

where:

*wd1* - WDF array number

*string* - new dataset comment for the array

**Note:** The CHA command, with the *clabel* keyword, also perform this function

Example:

Label array 2 as "Ion Current vs. Time"

```
LAB, 2, 'Ion Current vs. Time'
```

## LABEL

Add a label or text string to a plot using the cursor for placement. The location is determined with the cursor. XYOUTS is used for the label.

**format:** LABEL [, LABEL\_STRING] [, Xval] [, Yval] [, /RESET] [. \_EXTRA = extrastuff]

where:

- LABEL\_STRING* - String to be placed on a plot. If string is undefined, zero length, or not a string, then the user will be prompted for a string.
- Xval* - X-value for the label in data coordinates
- Yval* - Y-value for the label in data coordinates
- Note:** If missing or if keyword (reset) is set, then the value will be set interactively.
- RESET* - If set, the user will be expected to set the position of the label using the cursor.
- \_EXTRA* - Alignment, charsize, charthick, Text-Axes, color, ... Any valid keyword for XYOUTS.

**Example:**

Label a plot at a point to be determined with a string to be entered.

**LABEL**

Label a plot on the screen and save the values for a postscript plot.

labelstring = ' ' or labelstring = 0 ; since zero is not a string

< plot command >

LABEL, labelstring, x,y, /RESET

startpost

< plot command >

<Note: X and Y (set with the cursor) and the text string, entered on the terminal, are in variables.>

LABEL, labelstring, x, y

endpost

**LEGEND**

Prints a plot legend

**format:** LEGEND, string, LINESYLE = linestyle, PSYMBOL = psymbol, COLOR = color, POSITION = position, /device, /data, /normal, /relative, MSPACE = mspace, THICK = thick, NEXTPOS = nextpos

where:

- string* - String array with the legend text for each line in the legend.
- linestyle* - Array of integers indicating the linestyle for each line Default is -1, solid line. Other values are the same as IDL -1, No LINE, 0, Solid; 1, Dotted; 2, Dashed; 3, Dash-Dot; 4, Dash-Dot-Dot-Dot; 5, Long Dashes  
 Linestyle should have one value for all curves or one value for each value of string. If there are fewer values, then they will be recycled.

- psymbol* - Array of integers indicating the symbol for each line Default is 0, no symbol. If there are fewer values of *psymbol* than curves then they will be recycled. Values are the same as IDL (*abs(psymbol)* is used). 1, +; 2, \*; 3, period(.); 4, Diamond; 5, Triangle; 6, Square; 7, X; 8, UserDefined.  
(If the symbol value is greater than 10 then ll will be subtracted from the value and MKSYMBOL will be called. (See MKSYMBOL for a description of additional symbols.)  
**Note:** Symbols may also be specified as names:  
PSYM=['sq', 'ci', 'diamond', 'clover']  
These will be converted to numbers. [11, 12, 14, 21]  
A null or blank string('' or ' ') will result in no symbol
- color* - Color for symbols and lines for each line of the legend Default is !p.color (white for X and Black for PostScript) If there are fewer values of *color* than curves then they will be recycled. Values are: 1, red; 2, green; 3, blue; 4, magenta; 5, orange; 6, cyan; 7, yellow; 8, white; 0, black
- position* - Position on the plot in units determined by keywords.  
If POSITION is defined the default units are data coordinates.  
If POSITION is undefined the default is a relative position of [0.04, 0.94] of the plotting area.
- /device, /data,*  
*/normal, /relative* - only one may be specified. Determines the coordinates for the position keyword  
Data refers to the plot coordinates (Default)  
Relative is relative the actual grid  
Device is in device coordinates  
Normal is normalized space coordinates
- Mspace* - Multiplier for the spacing between lines
- Thick* - parameter sent to *oplot*
- NEXTPOS* - Next position for the legend in data coordinates  
Use this keyword for successive calls to legend
- Charsize* - parameter sent to *xyouts*

**Example:**

Label curves with numbers in the upper left hand corner of the grid

```
LEGEND, sindgen(10), pos= [0.04, 0.94], /relative, lin = 1, color = [2, 3, 4, 5, 6, 7, 1]
```

## LHELP

Provides a graphical user interface to the user help procedures `mk_library_help` can be used to generate the help file. `IDL_HELP_PATH` can be used to add user routines to the PFIDL routines.

**format:** LHELP [, /html] [, /remote] [, /set\_default] [, /show\_default]

where:

*html* - If set, an html version of help will be used.

Default = 0

*remote* - If set, a Netscape browser must be running.

Default = 0

Note: If either of both of the following keywords are set, then help will not be invoked.

*set\_default* - If set, the value of *html* and *remote* if present, will be set as the default.

*show\_default* - If set, show current defaults.

Examples:

Display an old style IDL help widget

```
LHELP
```

Limit array 2 to abscissa values of 0 to 200 ns.

```
LHELP, /remote. /html
```

## LIM

Limit the range or domain of a WDF array. Limits on the domain of the function are approximate; the domain is reduced to the first data value with abscissa  $\geq$  start and the last data value  $\leq$  stop.

**format:** LIM, wda, start, stop [, /YAXIS]

where:

*wda* - WDF array number or an array of array numbers

*start, stop* - Lower and upper limits for the array

*yaxis* - Nonzero value indicates the range of the function is to be modified

Zero or missing indicates the domain of the function is to be modified

Examples:

Limit array 2 to a range of y-values between 0 and 20.

```
LIM, 2, 0, 20, /y
```

Limit array 2 to abscissa values of 0 to 200 ns.

```
LIM, 2, 0, 200e-9
```

Limit arrays 2 to 5 to abscissa values between *begin* and *end*

LIM, [2,3,4,5], begin, end ; or: lim, indgen(4)+2, begin, end

## LINTR

Generate lineouts from any 2-D structure. Single block data generally uses the actual data for the lineout. Multiple block data is put on a uniform grid with 1 pixel resolution. Procedure will generate lineouts and display them. Lineouts use the existing grid in one direction. The X or Y values are used depending on which direction provides the most values between the designated end points. Lineouts must have a minimum of 3 values. Distances are slant distances from one end point. When designated, the lineouts will be stored in WDF arrays. Horizontal and Vertical lineouts require a single point. A preview gives lineout before storage. A preview will be generated whenever the mouse moves in the window and one point has been selected for a general lineout or if no points are selected for a vertical or horizontal lineout.

**Note:** Some time is required to calculate a preview frame. All events generated while the preview frame is generated are deleted; be sure the mouse clicks are acknowledged before proceeding.

Mouse buttons are mapped to widget buttons, cursor must be in the DRAW area of the main widget for the following mouse commands.

Left Button = Set Value

**Note:** A carriage return in the data field will also set the value.

Middle Button = Accept/Save (current designated lineout will be saved)

Right Button = Reset All Points (All points will be undesignated)

**Note:** If a the last point required to define a curve is selected more than once, the previous "last point" is replaced.

Set Point Buttons: These perform the same function as a <CR> in either the X or Y windows above the buttons.

**format:** LIMSTR, STR [, WDF] [, X, Y]

where:

*STR*

- Structure array numbers of a two dimensional structure or a slice from three dimensional structure. The structure should be single block for best results. Multi-block structures will be put on a uniform 256 by 256 grid. To use an interpolation with N elements use:

PLOSTR, STR, NFARRAY = N, RESULT = STR2

LINTR, STR2, wdf

**Note:** Data will not be transposed.

*WDF*

- WDF array number to store the lineout. Default is 1. WDF can be set manually.

*X, Y*

- N\*X values and N\*Y values if neither horizontal nor vertical is set. Lineouts of arbitrary pairs of

points may be obtained directly

or

- X* - N values corresponding to Y or X values if horizontal or vertical is set, respectively.

Keyword Parameters:

- Horizontal* - If present and not zero, then a full width horizontal lineout is taken. Can also be set manually. Only one point must be designated.
- Vertical* - If present and not zero, then a full height vertical lineout is taken. Can also be set manually. Only one point must be designated.
- Preview* - If present and zero, no preview will be generated. If positive, preview will have a fixed scale equal to the total range of data. If preview has two elements, these will be used as the y-range of the preview frame.  
Default is preview = -1 => Auto scale preview.  
**Note:** Preview frame may be resized.
- Width* - In current version, ignored unless a horizontal or vertical lineout is requested.
- WDFarray* - Array of WDF arrays where lineouts are stored. If no values are stored, then WDFarray = 0
- Xonly* - Return the abscissa values only not the length along the lineout, ie the wdfarray will be amplitude vs x axis value. Xonly is ignored if  $\Delta X / \max(X) < 1e-4$
- Yonly* - Return the ordinate values only not the length along lineout, ie the wdfarray will be amplitude vs y-axis value. Yonly id ignored if  $\Delta Y / \max(Y) < 1e-4$
- SLOPEAngle* - Angle is degrees of the slope of the last lineout. Result of arc tangent with two arguments.
- CPRINT* - If set, print the dataset comment for each lineout to the terminal
- \_EXTRA* - Any valid keyword for plotfld. The array for the lineout is generated with the following command and this plot is used for the image:  
PLOTFLD, str, nfarray = 256, xarray = x, yarray = y, farray = array, nlev = 29, \_extra = extrastuff  
**Note:** nfarray is ignored with single block data

Examples:

Take a series of lineouts from structure array 3 and store the lineouts

beginning in wdf array 1.

**LINSTR, 3, 11**

Take a series of lineouts from structure array 3 and store the lineouts beginning in wdf array 1; Set the preview to fixed scaling.

**LINSTR, 3, 11, /preview**

or

**LINSTR, 3, 11, /prev**

Take a series of lineouts from structure array 1 and plot the lineouts 12 to a page

**LINSTR, 1, wdf= wdf**

**plo, wdf, g =12**

Take a radial lineout from (0, 0) to (3, 3) and store the result in wdfarray 4 from structure 82

**LINSTR, 82, 4, [0, 3], [0, 3]**

## LOADXYCT

Loads the colors for an x, y plot.

**format:** **LOADXYCT, color\_table.**

**NOTE:** Load a color table and set the bottom 9 colors as shown below. If color table is not specified then the last color table specified will be loaded.

where:

<i>color</i>	- 0 black
	1 red
	2 green
	3 blue
	4 magenta
	5 orange
	6 cyan
	7 yellow
	8 white

## LOP

Apply a low pass filter to a WDF array.

**Note:** If *wdf2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wdf1*.

**Note:** See *FILT* for keyword definitions. This routine is called by *FILT*.

**format:** **LOP, wdf1 [, wdf2] [, TYPE = type] [, ATTEN = filter] [, ORDER = order] [, MODE= bin] [, /PLOT].**

where:

*wd1, wd2* - WDF array numbers  
 If *wd2* is present, then  $wd2 = \text{LOP}(wd1)$   
 else  $wd1 = \text{LOP}(wd1)$

Default keyword values are:

*type* = 'BUT'  
*order* = 2  
*bin* = 12

Example:

Apply a low pass filter to WDF array 1 and store the filtered array in array 2 using the default parameters

**LOP, 1, 2**

## LPRINT

Procedure to send a postscript file to a printer

**format:** LPRINT [, *file*] [, **DESTINATION** = *destination*] [, **COLOR** = *color*]

where:

*file* - file to be printed- If *file* is specified, then that value will become the default in future calls to LPRINT. To return to the default value, use ' ' for the file  
 DEFAULT = 'idl.ps'

**DESTINATION** - site specific. If **DESTINATION** is specified, **COLOR** is ignored. If **DESTINATION** is specified, that value will be used on all future calls to LPRINT. To return to the default value of the destination use **D=** ' Default = 'ljet' if **color** = 0. Default = 'pscolor' if **color** = 0.

**COLOR** - if present and not zero, indicates the default color printer. This parameter is ignored if **DESTINATION** is specified Default is **COLOR** = 0.

Examples:

Print idl.ps on the default printer

**LPRINT**

Print /users/lpuser/idl/junk.ps on the default printer

**LPRINT, '/users/lpuser/idl/junk.ps'**

**Note:** to print the same file again use the command: lprint

Print idl.ps on the printer pscolor

**LPRINT, /c**

or if all prints are to go to pscolor use

**LPRINT, d = 'pscolor'**

**LOP, 1, 2**

## LS

Show files in current directory

**format:** LS [, SPEC] [, CURRENT = current]

where:

*SPEC* - File filter. If SPEC is an array, only the first element is used.

Keywords:

*CURRENT* - Files in the current directory

Outputs:

CURRENT directory file names

Examples:

Show all files in the current directory

**LS**

Show all files which end if .dat

**LS, '\*.dat'**

## MAXSTR

Find the maximum of a structure array and the locations of the maximum. Current version is for type 1 field structures.

**format:** Field\_min = MAXSTR (structures [, vector])

where:

*Field\_min* - the minimum value of the field.  
*structures* - structure array number or IDL field structure  
*vector* - vector component for the function. If *vector* is an array, the square root of the sum of squares will be used. If *vector* is a single value array, ABS will be applied to that component. That is if *vector* = [2] then ABS(V2) will be searched. If max(vector) has a value larger than the number of vector components and dimensions. then *vector* = 1+indgen(nvect) where nvect is the number fo vector componenets.  
Default: Vector = 1 if number of vectors is 1  
Vector = 99 if number of vectors is > 1

Keyword Parameters:

*xrange* - Range of the first dimension for the minimum search.  
Default: All X values

*yrange* - Range of the second dimension for the minimum search.  
Default: All Y values

*zrange* - Range of the third dimension for the minimum search.  
Default: All Z values

<i>location</i>	- Position where maximum occurs
<i>pmaximum</i>	- Same as location
<i>minimum</i>	- minimum value in the range defined.
<i>pminimum</i>	- position where minimum occurs
<i>ier</i>	- error parameter Default: <i>ier</i> = 0 ==>no problems

## Examples:

Determine the range fo the electric field structure E

**range = MAXSTR(E, 4, min = emin) - emin**

**MINSTR**

Find the minimum of a structure array and the location fo that minimum. Current version is for type 1 field structures.

**format: Field-min = MINSTR(structure [, vector])**

where:

<i>Field_min</i>	- the minimum value of the field.
<i>strutures</i>	- structure array number or IDL field structure
<i>vector</i>	- vector component for the function. If <i>vector</i> is an array, the square root of the sum of squares will be used. If <i>vector</i> is a single value array, ABS will be applied to that component. That is if <i>vector</i> = [2] then ABS(V2) will be searched.If max(vector) has a value larger than the number of vector components and dimensions. then <i>vector</i> = 1+indgen(nvect) where nvect is the number fo vector componenets. Default: Vector = 1 if number of vectors is 1 Vector = 99 if number of vectors is > 1

## Keyword Parameters:

<i>xrange</i>	- Range of the first dimension for the minimum search. Default: All X values
<i>yrange</i>	- Range of the second dimension for the minimum search. Default: All Y values
<i>zrange</i>	- Range of the third dimension for the minimum search. Default: All Z values
<i>location</i>	- Position where maximum occurs
<i>pminimum</i>	- Same as location
<i>maximum</i>	- maximum value in the range defined.
<i>pmaximum</i>	- position where maximum occurs
<i>ier</i>	- error parameter Default: <i>ier</i> = 0 ==>no problems

## Examples:

Determine the range fo the electric field structure E

**emin = MINSTR(E, 4, max = emax)**

**range = emax - emin**

**MED**

Median filter of a WDF array. This filter is ideal for “salt and pepper” noise (isolated high and low values).

**Note:** If *wd2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**Note:** The median is the middle value; for the array [1, 2, 3, 4, 25], the median is 3.

**format:** **MED, wd1 [, wd2] [, WIDTH = width]**

where:

- |                 |   |
|-----------------|---|
| <i>wd1, wd2</i> | - WDF array numbers<br>If <i>wd2</i> is present, then <i>wd2</i> = MED ( <i>wd1</i> )<br>else <i>wd1</i> = MED ( <i>wd1</i> ) |
| <i>width</i>    | - width of the median filter; the default value is 5  |

## Examples:

Calculate the median filter (width = 5) of array 2 and store the result in array 3.

**MED, 2, 3**

**MID**

This function calculates the center of a wdf array according to the equation:

$$\text{center} = \text{total}(x \cdot dx \cdot y) / \text{total}(dx \cdot y)$$

If keyword cylindrical is set then mid returns:

$$\text{center} = \text{total}(x \cdot x \cdot dx \cdot y) / \text{total}(x \cdot dx \cdot y)$$

Procedure:

1. Get x, y, and calculate dx.  
DX calculated as average dx on either side if not uniform.  
End values of DX are the end values.
2. Get offset
3. Calculate MID = TOTAL(x\*dx\*(y-offset))/TOTAL(dx\*(y-offset))
4. TOTAL is the idl function

**format:** **center = MID(wdf [, LEFT = left] [, RIGHT= right] [, /PLOTIT] [, /QUIET] [, /INQUIRE] [, OFFSET = offset] [, AREA =area] [, IERR = ierr] [, /cylindrical]**

where:

- |               |  |
|---------------|--|
| <i>center</i> | - Center point defined above<br>If abs(area) is less than 10e-2*xrange*yrange/2) a warning will be issued. (where xrange = right-left and yrange = max(y) -min(y)) Perhaps less baseline or other modification will prove a better answer. |
|---------------|--|

<i>wdf</i>	- WDF array number or name of a wdf array
<i>LEFT</i>	- Minimum X value to consider. If left is outside the data range, it is set to minimum.
<i>RIGHT</i>	- Maximum X value to consider. If right is outside the data range, it is set to the maximum.
<i>INQUIRE</i>	- If set, waveform will be plotted and user asked to indicate the baseline.
<i>OFFSET</i>	- If inquire is set, the baseline will be returned in offset. If inquire is not set, then the value is offset will be subtracted from the ordinate values. Default = 0.0
<i>AREA</i>	- Value of Total(dx*y). For some shapes this is an area; for others, an integral.
<i>PLOTIT</i>	- If set the waveform, offset, and median will be plotted
<i>QUIET</i>	- If set, will not print the area warning.
<i>IERR</i>	- Set to 0 if data is valid, -1 if not valid. Set to 1 if area fails the above criteria.

Example:

Find the center to the following:

```
x = findgen(100) & y = x*exp(-x/20) & i2w, x, y, 1
```

```
center = MID(1. /pl)
```

Find the center of mass for waveform 4, using the average of the data from the start to 50 ns as the offset. Calculate the mid point using data between 50 and 150 ns.

```
cm = MID(4, left=50e-9, ri=150e-9, off=ave(4,r=50e-9, /qui))
```

Find the median of waveform 'lineout 3' and plot the result. Set an offset using a widget.

```
median = MID('lineout 3', /plot, /inquire)
```

## MIRROR

Mirror a WDF array about its initial point. Store the result in the initial WDF array or another WDF array.

format: **MIRROR, WD1, [, WD2]**

where:

<i>wd1</i>	- WDF array number of the first array
<i>wd2</i>	- WDF array number for the composite; if not specified the composite is stored in WD1 the first array

Examples:

Mirror WDF array 2 and store the result in WDF array 8

**MIRROR, 2, 8**

## MKSYMBOL

Make neat little user defined symbols. IDL routine USERSYM is used, so any keyword accepted by this routine can be used in MKSYMBOL.

**format:** MKSYMBOL, symbol, SIZE=size, LIST=list, HELP=help, \_EXTRA = extra  
where:

- |               |  |
|---------------|--|
| <i>Symbol</i> | - an integer indicating the desired symbol or a string with sufficient characters to uniquely identify the symbol. Current options are:<br>0, Square<br>1, Circle<br>2, Triangle<br>3, Diamond<br>4, Big X or X<br>5, Delta<br>6, Star<br>7, Plus<br>8, Hexagon<br>9, Pentagon<br>10, Clover<br>11, PACMAN<br>12, Spiral<br>DEFAULT: symbol = 1 (CIRCLE) |
| <i>SIZE</i>   | - Symbol size (default=1)  |
| <i>LIST</i>   | - using LIST=Some_variable will return the vector of available symbol names in Some_variable   |
| <i>HELP</i>   | - returns this documentation header  |
| <i>EXTRA</i>  | - <i>FILL, COLOR, THICK</i><br>See USERSYM documentation.  |

Examples:

Define a clover leaf for plot symbol 8.

**MKSYMBOL, 10 <or> MKSYMBOL, 'cl'**

Define a filled pacman for symbol 8

**MKSYMBOL, 11, /fill <or> MKSYMBOL, 'pa', /fill**

## MN

Function which returns the minimum, or local minimum, of a WDF array and optionally the abscissa value at which it occurs. Optional parameters allow the specification of a limited portion of the array. Default value is zero.

**format:** `min_value = MN (wd1 [, TIME = time] [, LEFT = left] [, RIGHT = right] [, QUIET = quiet])`

where:

- min\_value* - minimum value of WDF array between the interval specified
- wd1* - WDF array number
- time* - abscissa value for which the local minimum value occurs; for multiple minima, this is the first
- left, right* - optional abscissa value to specify the left and right abscissa values used to determine the local minimum
- quiet* - if present and nonzero, no values will be printed to the terminal

**Note:** The IDL function MIN is used to determine *min\_value* and *time*.

Examples:

Print the minimum value of array 2, between 10 and 1000

```
Print, MN(2, l = 10, r = 1000)
```

Calculate the minimum value of array 3 and store it in *bot* and the time at which it occurs in *btime*; suppress terminal output

```
bot = MN(2, time = btime, /quiet)
```

## MUL

Multiply a WDF array by another WDF array at corresponding abscissa values. Linear interpolation is used to put the arrays on the same abscissa values. The result only contains values in the region of overlap of the two arrays.

**Note:** If *wd3* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**Note:** *TSH* and *NEWEND* can be used to add zero values to the beginning and end of a WDF array.

**Note:** *CHA* can also be used to scale (multiply) data.

**format:** `MUL, wd1, wd2 [, wd3] [, CLABEL = clabel] [, XLABEL = xlabel] [, YLABEL = ylabel]`

where:

- wd1, wd2, wd3* - WDF array numbers  
If *wd3* is present, then  $wd3 = wd1 \times wd2$   
else  $wd1 = wd1 \times wd2$
- clabel* - Dataset comment for the sum  
Default is the dataset comment for *wd1*
- xlabel* - X-axis label for the sum  
Default is the x-axis label for *wd1*
- ylabel* - Y-axis label for the sum

Default is the y-axis label for *wd1*

Examples:

Multiply array 1 by array 2 and store the result in array 3

**MUL, 1, 2, 3**

Multiply array 3 by array 4 and store the result in array 3

**MUL, 3, 4**

Multiply each element of a WDF array by a constant.

**Note:** To multiply two constants, use IDL directly. (See last example.)

format: **MUL, wd1, 0, value [, wd3]**

where:

- value* - constant or first element of an array
- wd1, wd3* - WDF array numbers
- If *wd3* is present, then  $wd3 = wd1 \times value(0)$
- else  $wd1 = wd1 \times value(0)$

Examples:

Multiply all the elements in array 1 by 10 and store the result in array 3

**MUL, 1, 0, 10, 3**

Multiply all the elements in array 3 by the variable, *shift*, and store the result in array 3

**MUL, 3, 0, shift**

Multiply arrays 1, 2, 3 by array 4 and store the results in arrays 5, 6, 7

**for i=1, 3 do MUL, i, 4, 4+i**

Multiply variable, *width*, by variable, *length*, and store in *area*

**area = width \* length**

## MULSTR

Multiply a field structure array to another field structure array or to a constant. Store the result in the initial field structure array or another structure array. All attribute or vector quantities are multiplied.

**Note:** *CHASTR* can also be used to scale (multiply) data.

format: **MULSTR, STR1, STR2, [, STR3] [, CLABEL = clabel]**

format: **MULSTR, STR1, 0, VAR, [, STR3] [, CLABEL = clabel]**

where:

- STR1* - array number of the first field structure array
- STR2* - array number of field structure to be multiplied to first or 0, *VAR* Variable or constant to multiply the first structure
- STR3* - array number for the product, if not specified the

- product is stored in STR1 (the first array)
- CLABEL* - Optional label for the combined structures.
- RESTRICTIONS*
- STR1 array must be defined and have valid data
  - STR2 or 0 and a Variable must be defined
  - STR1 and STR2 must both be field arrays and have the same number of vector components.
  - The SIZE of STR1 and STR2 must be the same. The values of the spatial coordinates must be equal to about  $3e-5$  (ie.-  $\max(\text{abs}(\text{difference}))/\max(\text{value})$  It  $3.1e-5$  ).
  - The check on spatial coordinates is done globally rather than on a block by block basis.
  - If a structure has three dimensions, but is degenerate in one of them that axis need not be equal.
- PROCEDURE*
- If STR2 is specified Each attribute, or vector component, of STR1 is multiplied to each attribute of STR2 If 0, VAR is specified VAR is multiplied to each attribute of STR1 In either case, the product is stored in STR3, if specified, or STR1,if STR3 is not specified.*CLABEL* Optional label for the combined If 0, VAR is specified:

**EXAMPLES:**

Multiply structure arrays 2 and 3 and store the result in structure 8

**MULSTR, 2, 3, 8**

Multiply the variable OFFSET to structure array 2 and store the product in structure 8

**MULSTR, 2, 0, offset, 8**

Multiply structure arrays 3 and 8 and store the result in structure 10 with a dataset comment, 'Total POWER - Current \* Voltage'

**MULSTR, 3, 8, 10, c= 'Total POWER - Current \* Voltage'**

**MX**

Function which returns the maximum, or local maximum, of a WDF array and optionally the abscissa value at which it occurs. Optional parameters allow the specification of a limited portion of the array.

format: **max\_value = MX (wd1 [, TIME = time] [, LEFT = left] [, RIGHT = right] [, QUIET = quiet])**

where:

- max\_value* - maximum value of WDF array between the interval specified

- wd1* - WDF array number
- time* - abscissa value for which the local maximum value occurs; for multiple maxima, this is the first
- left, right* - optional abscissa value to specify the left and right abscissa values used to determine the local maximum
- quiet* - if present and nonzero, no values will be printed to the terminal

**Note:** The IDL function MAX is used to determine *max\_value* and *time*.

Examples:

Print the maximum value of array 2, between 10 and 1000

```
Print, MX(2, l = 10, r = 1000)
```

Calculate the maximum value of array 3 and store it in *top* and the time at which it occurs in *ttime*; suppress terminal output

```
top = MX(2, time = ttime, /quiet)
```

## MYTITLE

Write a general title for a plot at the top of the plot.

**format:** MYTITLE [, mytitle] [, OFFSET = offset] [, CHARSIZE = charsize] [, /BOTTOM] [, \_EXTRA = extrastuff]

where:

- mytitle* - Title for this plot. This is required.
- OFFSET* - This is added to the default position. Units are normalized units. If offset is a single value, it will be applied to the vertical position.
- CHARSIZE* - Character size.  
Default is 1.2.  
If maximum (!p.multi(1:2)) gt 2 then the size is scaled by 0.
- BOTTOM* - If set, text will be put below the plot.
- \_EXTRA* - Any valid keywords to XYOUTS

Procedure:

XYOUTS is used to print a general text string. Position is  $0.5 * \text{total}(!x.\text{window})$  in the horizontal direction and  $0.5 * (!y.\text{window}(0) + !y.\text{region}(0))$  in the vertical direction (Bottom) or  $0.5 * (!y.\text{window}(1) + !y.\text{region}(1))$  in the vertical direction (Top)

Examples:

**Note:** Before writing a title, a plot must be made.

Put the title 'X-Ray Analysis' on a plot and make it the default.

**MYTITLE, 'X-Ray Analysis'**

Print the title 'Special Title' with character size = 2

**MYTITLE, chars = 2, 'Special Title'**

## NEWEND

This procedure adds zero value points to a WDF array to extend its domain to a specified value. If the array's domain already includes the specified endpoint, no action is taken.

**Note:** To add zero values to an array before the beginning of the data, use the TSH command.

**format:** NEWEND, wd1, value

where:

- wd1* - WDF array number
- value* - designated final abscissa value

Examples:

Extend the data in array 1 by adding zero values to an abscissa value of 200e-9

**NEWEND, 1, 200e-9**

Extend the data in array 2, presently with values between abscissa values of 65e-9 to 180e-9, to cover the domain [0, 200e-9] by adding zero values at both ends

**tsh, 2**

**NEWEND, 2, 200e-9**

## OPENPFF

Open a PFF file. The current file pointer is set to this file id and the dataset pointer for this file is set to 1.

**format:** OPENPFF [, file] [, fileid] [, PATH = path] [, FILTER = filter]

where:

- file* - name of the PFF file to be opened  
The default file extension is ".PFF". Both upper and lower case versions of the file name will be tried.  
If file is omitted, then the IDL routine, *PICKFILE*, will be used to select a file. Keywords, *path* and *filter*, are passed to *PICKFILE*.
- fileid* - user specified file id  
If *fileid* is specified, it will be used as the file id. If *fileid* is not provided or is zero, a value will be provided; default value is the first unused integer  $\leq 1$ . If *fileid* is provided but has been used for another

file, the program will inquire if the user wishes to close the old file, to specify a new file id, or to accept the default.

*path, filter*

- see IDL help PICKFILE

**Note:** PICKFILE has been modified to return to the last directory accessed.

#### Examples:

Open a PFF file with the default file id. Select the file from a menu.

**OPENPFF**

Open a PFF file, '/users/test', with the default file id

**OPENPFF, '/users/test'**

Open a PFF file in directory '/extra/qs/' with extension '\*.pff'

**OPENPFF, PATH = '/extra/qs' , FILTER = '\*.pff'**

Open a PFF file with a file id of 15

**OPENPFF, ' ', 15**

Open a PFF file, '/users/test' with a file id of 14

**OPENPFF, '/users/test', 14**

## PAD

Decrease the point spacing of a WDF array using a FFT technique. This technique attempts to preserve the frequency content of the waveform. A fft is taken and zeros are added at higher frequencies to bring the inverse point spacing to the desired value.

Note: If the waveform does not begin and end at the same value of the amplitude then distortions will be generated at both extremes.

**format: PAD, WD1, DDT**

where:

*WD1* -WDF number or an array of numbers of WDF arrays to be changed

*DDT* - Decrease the interval between points by this factor. Default is 2. If DDT is less than or equal to 1, the routine returns with no operation.

#### Examples:

Increase the number of points in dataset 22 by a factor of 3

**PAD, 22, 3**

Increase the number of points in datasets 1 to 10 by a factor of 4

**PAD, 1+indgen(10), 4**

**PARSER**

Convert a string to a float or integer array. A space is assumed to be the normal delimiter.

**format:** PARSER, string, array, np [, INTEGER = integer] [, IGNORE = ignore]

where:

string	-input string to be separated
array	- output array
np	- number of elements in array
INTEGER	- if set, array is converted to an integer with ROUND
IGNORE	- any nonnumeric characters to be ignored or taken as delimiters can be listed in the string array, IGNORE. (See the example.)

Examples:

Parse a string, STR, into an integer array, array, and return the number of points found in, NP.

```
PARSER, str, array, np, /INTEGER
```

Read all remaining records in a file , UNIT, and put the numbers in NEW. Ignore “,” , “\$”, and “&”.

```
npmax = 4096
new = make_array(npmax)
i = 0L
str = ‘ ‘
IGNORE = [“,”, “$”, “&”]
while (not eof(unit)) do begin
  readf, unit, str
  PARSER, str, array, np, IGNORE = ignore
  if i+np ge npmax then begin
    npmax = npmax + npmax
    new = [new, new]
  endif
  if np gt 0 then new (i:i+np-1) = array(0:np-1)
  i = i+np
endwhile
if i gt 0 then new = new(0:i-1) else new = 0
```

**PE2PFF**

Convert a Perkin Elmer file to a pff file

**format:** PE2PFF [, File\_Name] [, IMAGE] [, SPACE] [, TLABEL = tlabel]  
[, CLABEL = clabel] [, BLABEL = blabel] [, XLABEL = xlabel]

```
[, YLABEL =ylabel] [, HEADER = header] [, NPOINTS = npoints]
[, DELTA =delta] [, OUTFILE =outfile] [, FILTER =filter] [, SCALE = scale]
[ SWAP = swap][, PATH = path] [, DELETE = delete] [, XIMAGE= ximage]
[XHEADER = xheader] [, PHEADER = pheader] [, DESTINATION = dest]
[, PLOTIt = plotit]
```

where:

- |                  |  |
|------------------|--|
| <i>File_Name</i> | - the name of the file with or without an extension if no name is specified, PICKFILE will be used   |
| <i>IMAGE</i>     | - array containing the image file  |
| <i>SPACE</i>     | - array containing the dataset x and y arrays. Use SP2XYZ, image, space, x,y to obtain x and y (SPACE = [X, Y])<br><b>Note:</b> MicroD offsets are ignored. Both X and Y start at (0, 0)   |
| <i>TLABEL</i>    | - Dataset type label. Default is the value of "SETVAL" (Specular Density, Transimission of Intensity)  |
| <i>CLABEL</i>    | - Dataset comment label. Default is File Name // Comment1 // Comment2 // ScanTime  |
| <i>BLABEL</i>    | - title or label for this block. Default is Specular Density, Transmission or Intensity depending on the value of "SETVAL"   |
| <i>XLABEL</i>    | - horizontal dimension label for this block. Default is 'X (microns)'  |
| <i>YLABEL</i>    | - Vertical dimension label for this block. Default is 'Y (microns)'  |
| <i>HEADER</i>    | - Complete header for the PE dataset   |
| <i>NPOINTS</i>   | - Array of points for each dimension   |
| <i>DELTA</i>     | - Array of microns/step for each dimension   |
| <i>OUTFILE</i>   | - File name without path information   |
| <i>FILTER</i>    | - Optional filter. If specified this parameter must have at least two values which are taken as the maximum and minimum of the scanned data. If three parameters are given, the third is taken as the default value given to all data which falls outside the specified range. Default value is [0, 5000, -1] . That is all data less than 0 and greater than 5000 will be set to -1. With the default scaling of 800, this corresponds to values between 0 and 6.25 |
| <i>SCALE</i>     | - Parameter used to scale the integer counts. Default is $1/800 = 1.25e-3$   |
| <i>SWAP</i>      | - This keyword is set by default, ie swap = 1. If swap = 0, then the bytes in image will not be swapped.   |

- Note:** VAX data must be swapped.
- PATH* - Directory path to the desired file. This is the same default value as used in openpff
- DELETE* - If set, delete the Perkin Elmer files
- XIMAGE* - Extension for the image file. Default is .Img
- XHEADER* - Extension for the header file. Default is .Hdr
- PHEADER* - If the keyword is set, the header will be printed to the screen.
- DESTINATION* - Path of the destination file. If no destination is specified the file will be written in the same directory as the PE file
- PLOTIT* - If set, a crude plot will be made of the image

## Examples:

PE2PFF, 'S6411n5j' ; read file S6411n5j from current directory  
 or  
 PE2PFF; read a file to be determined from PICKFILE  
 or  
 PE2PFF, /pheader,'S6411n5j' ; read S6411n5j and print header  
 PE2PFF, /plot ; read a file to be determined and plot it

**PFF\_DIAG**

This routine sets pff diagnostic options

format: PFF\_DIAG [, FILE] [, DEBUG = debug] [, VERBOSE = verbose]  
 [, INQUIRE = inquire] [, /HELPME] [, /QUIET]

where:

- FILE* - Used only if debug is set. If debug is a string, FILE is ignored.
- DEBUG* - If set, DEBUG diagnostics are toggled. File must be provided unless debug is a string.
- VERBOSE* - If set, VERBOSE diagnostics are toggled
- INQUIRE* - Returns a two element vector with the value of 0 or 1 on whether verbose and debug are set.  
 INQUIRE(0) = 0 if verbose is not set  
               = 1 if verbose is set  
 INQUIRE(1) = 0 if debug is not set  
               = 1 if debug is set
- HELPME* - If set, returns the command format
- QUIET* - If set, no output to the terminal unless errors are present.

Examples:

Determine the status of PFF diagnostics

**PFF\_DIAG**

or

**PFF\_DIAG, inquire = inquire**

Toggle VERBOSE diagnostics

**PFF\_DIAG, /verbose**

Toggle DEBUG diagnostics with output to go to '/tmp/ debug.out'

Return the status of the diagnostics in stat

**PFF\_DIAG, '/tmp/debug.out', /debug, inquire = sta**

or

**PFF\_DIAG, debug = '/tmp/debug.out', inquire =stat**

Turn off debug diagnostic.

(**Note:** If debug is off, a message will be printed but debug will remain off since no file is provided.)

**PFF\_DIAG, /debug**

## PFF\_INFO

This routine gets pff file information

**format:** **PFF\_INFO** [, **FID**] [, **FNAME = fname**] [, **NDSET = ndset**] [, **DPOINT = dpoint**]  
[, **CURRENT = current**] [, **NFILE = nfile**] [, **FID =fid**] [, **QUIET = quiet**]

where:

**FID** - file id of pff file for which information is desired. If FID is undefined or zero, data for all files is returned.

**Note:** Only the positional parameter, FID, will be used to specify pff files. (see second example) FID (positional parameter) is not modified.

**FNAME** - array of file names (maximum of 160 characters)

**FID** - keyword used to return an array of file ids

**NDSET** - Number of datasets in each file

**DPOINT** - Dataset pointer location

**CURRENT** - Integer array = 0 for all files except Current file = 1 or if an error Current = -1 i error of the PFF\_INFO

For example: current = [0, 0, -1, 1] indicates:

First two files had good data but are not current

Third file had a problem with PFF\_INFO

Fourth file is current file

**NFILE** - Integer, number of open files

**QUIET** - If present and not zero, then no data will be printed

Examples:

Determine the number of open files and other pff parameters

**PFF\_INFO**

Get arrays of the fids and file name for all open files in arrays fid and name, but print nothing to the screen. ( Since the positional parameter is undefined, all file information is returned)

## PFF\_REGRID

Do density-to-exposure unfold and regridding of scanned PFF image file, and write out a new (smaller) PFF file containing the regridded exposure values. Used as a preprocessor to deal with high-res scanned data to avoid bogging down Vida.

format: **PFF\_REGRID**, *INFILE* [, *FITFILE*] [, /*ZERO\_ORIGIN*] [, *REGRID = regrid*] [, *XREGRID = xregrid*] [, *YREGRID = yregrid*] [, *OUTFILE = outfile*]

where:

- INFILE* - string containing PFF filename to process. User should also provide the extension (.pff, .PFF, etc) Enter 0 or ' ' to use pickfile.
- FITFILE* - string containing film response filename to use. Enter 0 or ' ' to use pickfile. Should have been created in WDFIT3 Can be pre-loaded with READWD3. Values read will be stored in the WDFIT3 common block. If omitted then current values in the WDFIT3 common block are used.

Keywords:

- ZERO\_ORIGIN* - if nonzero, specifies that x,y axis values should be offset so that lower left corner of image will be at 0,0.
- REGRID* - integer  $\geq 1$  to specify what factor to regrid data by.  
**Note:** that idl rebin routine is used - averages points. May lose at most regrid-1 of the uppermost rows or columns of scanned data in this process. Equivalent to specifying *x\_regrid=regrid* and *y\_regrid=regrid*.  
Default value: *REGRID* = 5
- X,YREGRID* - integers  $\geq 1$  allowing independent regridding factors in the x and y dimensions.  
Default value: *X,YREGRID* = *REGRID*
- DEFAULT\_VAL* - Default values for regrid if infile values are outside the range.  
DEFAULT = [0, value at VALID(1)]

Side effects:

Creates a new PFF file in current directory, with a '\_ur' suffix added to root

of filename. Can take considerable time on a large file

#### Restrictions:

User should NOT have a file of the same name as the output file in the working directory! Output filename will be modified.

#### Examples:

Fit film response data and unfold a pff file and write it out with a reduction in size of  $16 = 4*4$ , put (0, 0) at lower left corner

```
revida, 0, 1          ; read a vida output file
wdfit3, 1, /ylog     ; fit the data to a polynomial
PFF_REGRID, 0, regrid = 4, /zero
```

## PIXPLOT

Read an ascii file of amplitude, x and y values and optionally delta-X and delta-Y values. Generate a field type structure of the data. Blank lines are ignored and lines with less than 3 numbers are ignored. TQEDIT can be used to modify/add/delete regions. Alternatively, if the plotonly keyword is set a contour pixplot will be generated with the data, but no structure will be returned.

**Caution:** The TQBUILD code suite is used. Current data in this common block can be lost.

```
format:  str = PIXPLOT ([, fname] [, SKIP = skip] [, IGNORE = ignore] [, FAST = fast]
           [, CIRCLE = circle] [, XDELTA = xdelta] [, YDELTA = ydelta] [, NX = nx] [, NY = ny]
           [, NP = np] [, XLOG = xlog] [, YLOG = ylog] [, ZLOG = zlog] [, XRANGE = xrange]
           [, YRANGE = yrange] [, DEFAULT = default] [, HIGHRES = highres]
           [, XLABEL = xlabel] [, YLABEL = ylabel] [, CLABEL = clabel] [, TLABEL = tlabel]
           [, PLOTONLY = plotonly] [, LEVELS = levels] [, CCOLOR = ccolor]
           [, THERMOMETER = thermometer] [, TPARAM = tparam] [, NLEVEL = nlevel]
           [, _EXTRA = extrastuff] )
```

where:

- str* - PIXPLOT data is returned in structure, STR If an error occurs, STR = 0
- FNAME* - file name or a unit number. If a number then the file is read directly. If unit is equal to zero of fname is undefined then pickfile will be used to determine a file. If pickfile is called then fname will contain the file name on return from pixplot file should contain data of the form: amplitude xvalue yvalue or optionally (where absolute values are used for the last 2): amplitude xvalue yvalue deltax deltay
- SKIP* - If the file has several lines of header information which must be skipped, set skip equal to this number of lines. (Default: SKIP = 0)
- IGNORE* - String array. Space is used as a delimiter. Any nonnumeric characters to be ignored are listed in

- ignore. See example.
- FAST* - If set polyfill will be used to determine values; slight error can occur
- CIRCLE* - If set, an ellipse is drawn around the specified points  
Default is *CIRCLE* = 0. A rectangle is drawn
- XDELTA* - Default deltax value if not listed in the file  
Default is 0.5; must be greater than zero
- YDELTA* - Default deltax value if not listed in the file  
Default is 0.5; must be greater than zero
- NX, NY* - Number of resolution elements in the X and Y directions  
Default =  $np > 3$
- NP* - Default for nx and ny if not specified.  
Default value is 250 or 1250 depending on *HIGHRES*
- HIGHRES* - If set, Default value of np is 1250 otherwise np = 250. If highres is set then fast is set. If np is specified, this keyword is ignored.
- LOWRES* - If set, Default value of np is 50 otherwise np = 250. If np is specified, this keyword is ignored.
- XRANGE* - Range of x-Values
- YRANGE* - Range of y-Values
- DEFAULT* - Default value for the two dimensional array  
Default = 0.0, unless *zlog* is set
- XLOG* - If set, X-data is the log of the actual data
- YLOG* - If set, Y-data is the log of the actual data
- ZLOG* - If set, Z-data is the log of the actual data
- XLABEL* - X-label for the data - Default = 'X Axis'
- YLABEL* - Y-label for the data - Default = 'Y Axis'
- CLABEL* - Comment label for the dataset - Default = 'PixPlot Data'
- TLABEL* - Type label for dataset - Default = 'PixPlot Data'
- PLOTONLY* - If set, no structure is returned but the PixPlot data is plotted in a contour plot.

The following keywords are used only for *PLOTONLY*

- NLEVEL* - Number of contour levels. Default is 20
- LEVELS* - Contour level values, these will be sorted. If levels has < 2 elements, the actual values will be returned. If levels has more than 1 value, these values will be used for the contour plot and *nlevel* will be set to

- this number  
 Default levels:  
 $levels = v_{min} + (v_{max} - v_{min})/n_{level} * findgen(n_{level})$
- CCOLOR** - C\_color keyword values for the contour plot. If ccolor has < 2 elements, the actual values will be returned. If ccolor has more than 1 value, these values will be used for the contour plot with the c\_color keyword.  
 Default levels:  
 $ccolor = 9 + round((!d.table\_size - 10) * (1 + indgen(nlevel)) / nlevel)$   
**Note:** If Ccolor is passed and levels are passed, both will be sorted
- THERMOMETER** - If keyword is set a thermometere will be generated with the contour plot. Default IDL margins are [10, 3] for x and [4, 2] for y. If TPARAM does not have a Tag Name beginning with 'CU' then:  
 If no tag names begin with 'H' xmargin will be set to [!x.margin(0), 10] otherwise ymargin will be set to [8, !y.margin(1)]  
**Note:** xmargin or ymargin passed to plotfld will override these settings  
**Note:** If THERMOMETER is a structure, then TPARAM is ignored and set equal to THERMOMETER
- TPARAM** - Structure containing data for the thermometer, Tag names must be valid for the THERMOMETER procedure.
- \_EXTRA** - Any valid keywords for the plot command  
 Default values for the contour plot are:  
 plot, x, y, /nodata, xrange = xrange,  
 yrange = yrange, title = clabel, xlog= xlog, ylog = ylog, xtitle = xlabel, ytitle =ylabel,  
 \_extra = extrastuff

**Procedure:**

1. Calculate a center and a radius for x and y

if a log plot:

$$center = \sqrt{(x+dx)(x-dx)}$$

$$radius = \log((x+dx) / center)$$

for a linear plot:

$$center = x$$

$$radius = dx$$

2. Generate the region:

circle set: tqcond, centerx+radx\*cos(angle), centery+rady\*sin(angle)  
 where angle is an array of angles between [0, 2pi]

circle not set:

`xx = [centerx-radx, centerx+radx, centerx+radx, center-radx,  
centerx-radx]`

`yy = [centery-rady, centery-rady, centery+rady, center+rady,  
centery-rady]`

`tqcond, xx, yy`

3. edit the regions then fit to a grid and edit again if necessary fill the regions and construct a structure.

Example:

Assume a data file containing:

1 6 6 5 5

2 6 6 4 4

3 6 6 3 3

4 6 6 2 2

5 6 6 1 1

make a pixplot of this data - several plot options exist

`str = PIXPLOT(ignore = ';', xran = [0, 12], yran = [0, 12])`

`plotstr, str, /shade ; shade surface of the result`

`plotstr, str, /fill ; filled contour plot`

`plotstr, str, /fill, col =-1, nl = 5 ;RGB type contour plot`

Make a pixplot with full grid with file 'junk.dat' and set background to 3

`junk = PIXPLOT(ignore = ';', xran = [0, 12], /plot, 'junk.dat', ticklen = 1, /xgridstyle,  
/ygridstyle, default = 3)`

## PLO

Plot a single WDF array or multiple WDF arrays. The LEGEND routine is used to generate a plot legend for overlay plots.

**format:** `PLO, wda [,ncurve] [, /OVER] [, X RANGE = xr] [, Y RANGE = yr]  
[, GRID = grid] [, LEGEND = leg] [, NSYMBOL = nsymbol] [, /NOGRID]  
[, COLOR = color] [, PSYMBOL = symbol] [, L INESTYLE = line]  
[, /NL INESTYLE] [, DATA = data] [, TITLE = title] [, LGSP= legspace]  
[, XTITLE = xtitle] [, YTITLE = ytitle ] [, NCHAR = nchar]  
[, NEXTPOS = nextpos] [, THICK = thick] [, CHAR SIZE = charsize]  
[, LCHAR SIZE = lcharsize] [, ALLSYMBOL = allsymbol] [, UNSET = unset]  
[, SETDEFAULT = setdefault] [SHOWDEFAULT = showdefault ]  
[, HEADER = header] [, LEFT = left] [, RIGHT = right]  
[, ENGINEERING = engineering] [_EXTRA = extrastuff]`

where:

- `wda` - WDF array number or an array of numbers
- `ncurve` - number of curves to be plotted. If `wda` is a single `wda` array number, then `wda` and the following `ncurve-1` arrays will be plotted.

- color*
- indicates the color to be used for the curve. If only the keyword (/C) is present and the OVER keyword is specified then the successive curves will be drawn in different colors. If color is set to a value other than 1, then the first color will be given by this value and the value will be incremented by 1 for each additional color, that is :  

$$\text{color} = (\text{color} + \text{indgen}(7)) \bmod 7 + 1.$$
The colors are: 0 = black, 1 = red, 2 = green, 3 = blue, 4 = magenta, 5 = orange, 6 = cyan, 7 = yellow, and 8 = white. If color is an array, those values will be used for the overlay plot. If a color is set to zero, then !p.color is used for all curves.
- data*
- if present and not zero, indicates *leg* is in data coordinates.
- grid*
- integer indicating the number of plots to a page or an array indicating the number of plots in the horizontal and vertical directions. When grid is an array, the first element is the number of plots in the horizontal direction and the second element is the number in the vertical direction. When grid is a single number, N, then the number in the y-direction is given by:  $y = \text{fix}(\text{sqrt}(N))$  and the number in the x-direction is given by:  
 $x = \text{fix}(N / \text{float}(y) + 0.99999).$
- legend*
- location of the legend for overlaid plots. This is a two dimensional array which specifies the x and y location as a fraction of the grid dimensions. The default values are (0.04, 0.94) relative to the grid. If *leg* = -1, no legend will be drawn. *leg* may be specified in data coordinates using the keyword, *data*.
- lgsp*
- Scale the spacing of the legend by the absolute value of this parameter
- linestyle*
- Indicates the type of line to be used. Lines specified by LINESSTYLE are: solid(0), dot(1), dash(2), dot-dash(3), dot-dot-dot-dash(5), and long dash(5). If the keyword OVER is present then if LINESSTYLE contains:
    - a. more than one value, then the linestyle types are sequenced over the specified values.
    - b. a single value not equal to -1, then the linestyle types ; are given by  $\text{LINESYYLE} = (\text{LINESSTYLE} + [0,2,3,4,5,1]) \bmod 6$
    - c. If LINESSTYLE is -1, a solid line is used for all

- plots.  
If the keyword OVER is not present, then all plots are done with LINESTYLE (0) unless an array is specified.
- nlinestyle* - If present and nonzero, all the lines will be solid  
This is the same as linestyle = -1
  - nogrid* - indicates the curve should be plotted without drawing a new grid if present and nonzero.
  - npoint* - indicates the number of points used for XY data. Default is 500. This has no effect on XY data.
  - nsymbol* - indicates no symbols to be plotted on the curves in the overlay mode if present and nonzero.
  - allsymbol* - If present and not zero, a symbol is plotted at every point. If allsym is negative, only symbols will be plotted. If allsym is positive, a line will be drawn with a symbol at every point
  - nchar* - Number of characters in the legend
  - over* - indicates the curves are to be overlaid. A legend is printed to distinguish the curves. Line types and symbols will be used automatically. The location of the legend can be specified with the array LEG.
  - psymbol* - indicates the type of line and symbol to be used. If it is less than zero, a symbol will be plotted at every point and no line will be drawn. If *symbol* is greater than zero only about 40 symbols will be plotted and lines will connect the points. Values of *symbol* are: 1 is plus sign, 2 is asterisk, 3 is period, 4 is diamond, 5 is triangle, 6 is square, 7 is "X", and 10 is histogram mode with horizontal and vertical lines connecting the points. If the keyword *over* is present, then *symbol* is ignored and the symbol styles are sequenced over the datasets.
  - title* - string to be used as the title of the plot. If this is not present, the dataset comment of the first curve plotted on a grid will be used for the title. If overlay is set, then no title will be printed unless specified.
  - xr, yr* - two element arrays with the first element equal to the minimum and the second element, the maximum value of the x-axis and y-axis respectively. These values will be rounded to "nice" values unless *xstyle* and/or *ystyle* have been set.
  - xstyle, ystyle* - integers to specify the axis style. Setting bit 0 (a value of 1 or /XS (/YS)) forces exact axis range. Bit 1 (value 2) extends the axis range. Bit 2 (value 4)

- suppresses the entire axis. Bit 3 (value 8) suppresses a box style axis. Bit 4 (value 16) inhibits setting the Y axis minimum value to zero (Y axis only).
- xtitle, ytitle* - strings to be used for the X and Y axis labels.
- xtype, ytype* - specify logarithmic axis if present and nonzero
- nextpos* - specify the next line of output in data coordinates for additional calls to LEGEND
- thick* -Line thickness, default is !p.thick
- charsize* - Parameter sent to the PLOT command to vary the character size. If !P.CHARSIZE is not equal to zero (the default value), then charsize is a multiplier for !p.charsize. This is also sent to the LEGEND command with the multiplier, LCHARSIZE. Default is CHARSIZE = 1.
- lcharsize* - Parameter sent to the LEGEND command to vary the character size in the legend. The value sent is actually LCHARSIZE\*CHARSIZE. Default is LCHARSIZE = 1.
- engineering* - If set, the maximum axis labels will be in the range1 to 1000.
- left* - Indicates a new plot using the left axis. A single axis will be drawn on the left side of the grid. For this plot XMARGIN will be set to XMARGIN = [!X.margin(0), !X.margin(0)] and YSTYLE = 8.  
**Note:** If LEFT is set, NOGRID is set to 0.  
**Note:** If LEFT is set, OVER is set to 1 if more than 1 curve.
- right* - Indicates a NOGRID plot using the right axis. If RIGHT is present and not zero, then:  
 1. A single axis will be drawn on the right side of the existing grid  
 2. The wdf array(s) will be overplotted on the existing grid.  
 Before using this command, a plot should be created using the keyword: /LEFT.  
**Note:** If RIGHT is set then NOGRID is set to 1.  
**Note:** After this command all memory of the original y-axis scaling is lost.
- unset* - If present and not zero, will set all save values of keywords to their default values. This keyword can be used alone or in a regular plot command with other plot values or keywords including the SETDEFAULT keyword.

Examples:

Set the above defaults:

plo, /unset

Set the above defaults, make over = 1 the new default and plot curves [1,2,3] in an overlay plot:

plo, 1, 3, /over, /set, /unset

*setdefault*

- If present and not zero, then the following parameters will be saved and used as the default values in all future calls to PLO. If a keyword is specified, it will always be used rather than the saved values. This keyword may be used with a regular plot command; for example:

PLO, 1, 3, /over, /ns, /nl, /co, xr = [0, 150e-9], /set will set default values of 1 for color, nlinestyle, over, xrange and nsymbol. To temporarily change use the keyword; for example:

PLO, 1, 3, xr = [20, 80]\*1e-9

will use the above keywords but xrange would now be 20 to 80ns. The default value remains 0 to 150 ns until a new value is set or until PLO is called with UNSET. To temporarily use auto scaling for X: plo, 1, 3, xr = 0

Normal default values are listed below.

Keyword	Default value
COLOR	0
OVER	0
XRANGE	0
YRANGE	0
LINESTYLE	[0,2,3,4,5,1]
(LINESTYLE= -1 implies NLINESTYLE= 1 )	
GRID	0
PSYMBOL	[1,2,4,5,6,7]
LEGEND	[0.04, 0.94]
NCHAR	0
(NCHAR = 0 implies using the default of 30/12)	
DATA	0
ENGINEERING	0
NSYMBOL	0
ALLSYMBOL	0
CHARSIZE	1
LCHARSIZE	1

*showdefault*

- Show the present default plot values.

*header*

- If present and not zero, a plot header, consisting of the current date and the current file name, is

displayed in the lower left corner of the plot. See the HEADER command to set default HEADER parameters (character size, spacing, alignment).

DEFAULT is HEADER = 1

**Note:** For X-windows screen dumps, many will want to increase the character size with the command:

HEADER, /set, charsize = 0.9

(Default is 0.6 of current size)

*extrastuff* - any other valid keyword for the IDL PLOT command may be entered

Examples:

Plot array 1 using the default settings.

**PLO, 1**

Plot array 2 using diamond symbols and a dashed line over the existing graph.

**PLO, 2, s = 4, LINE= 2, /NOGRID**

Plot arrays 1,..., 6 using different colored lines on a single grid. Use the specified labels for the plot.

**PLO, 1, 6, /OVER, /COLOR, TITLE = 'Voltage at 6 Locations', XTITLE='Time (s)', YTITLE='Voltage (MV)'**

Plot array 1 in a log-log plot with exact axis scaling in the X direction.

**PLO, 1, X RANGE = [1e6, 12e6], /xstyle, /xtype, /ytype**

Plot an overlay of arrays 1,..., 6 with abscissa of 0 to 100 and range of 0 to 10

**PLO, 1,6, X RANGE = [0, 100], Y RANGE = [0, 10], /OVER**

Plot arrays 1,..., 10 on separate grids with 6 grids to a page

**PLO, 1, 10, GRID=[3,2] ; or: PLO,indgen(10)+1,g=6**

Plot arrays 1, 4,5, 9 on one grid

**PLO, [1,4,5,9], /OVER**

Plot all arrays beginning with "mocam" on a common time base. This requires two commands. Plot 4 horizontal and 3 vertical grids to a page.

**re, 1, '^mocam', nc = ncurve, /all**

**PLO, 1, ncurve, X RANGE = [0, 200e-9], GRID = [4, 3]**

## PLOTCON

Plot conductor geometry

format: **PLOTCON, A [, X] [, Y] [, ZVALUE] [, YRANGE = YRANGE] [, THICK =THICK] [, X RANGE = X RANGE] [, COLOR = COLOR] [, TITLE = TITLE]**

[, XTITLE = XTITLE] [, YTITLE = YTITLE] [, OVERPLOT = OVERPLOT]  
 [, \_EXTRA = extrastuff]

where:

- A - IDL particle structure (type = 4)
- X - Abscissa values (1, 2, or 3 x, y, or z coordinates)
- Y - Ordinate values (1, 2, or 3 x, y, or z coordinates)
- ZVALUE - Value of the coordinate in the third dimension. In the case of two dimensional data, enter 0.

**Note:** If overplot is not zero, then x and y, zvalue will be determined from the previous plot.

**Warning:** Slant surfaces are assumed to be two dimensional and are not checked to verify that they contain the appropriate ZVALUE.

KEYWORD PARAMETERS:

- OVERPLOT - If not zero, indicates no new grid is to be drawn.
- THICK - Line Thickness default is 2
- COLOR - All lines will be drawn with a single color. The colors are: black(0), red(1), green(2), blue(3), purple(4), orange(5), light blue(6), yellow(7) and white(8).
- TITLE, XTITLE, YTITLE - If OVERPLOT is not zero, these will be used for axis and the plot labels. If these are not string arrays, the title, xtitle and ytitle used for the plots will be used.
- X(Y)RANGE - Axis range (standard IDL meaning)
- \_EXTRA - additional parameters to the plot command. POSITION,

(X,Y)STYLE, (X,Y)TYPE have the standard IDL meaning.

EXAMPLE:

Plot from structure a, space 1 vs. space 3 use a grid [0,0.064, 0, 0.14]

```
PLOTCON, a, 3, 1, 0, XRANGE = [0,0.064], YRANGE = [ 0, 0.14]
```

Overplot conductor geometry from structure a

```
PLOTCON, a, /over
```

## PLOTcube

Plot a WDF array or structure array in a 3D cube. Can be used with scalar field arrays or wdf arrays. This routine can also be used to indicate a series of slices of a 3D shape. The default axis is z vertical, x horizontal and y into the screen.

**Note:** For postscript !p.font = -1 gives better results usually.

format: PLOTcube, NORMAL, ZVALUE, STR [, REVERSE = reverse [, SLICE = slice]  
 [, WDFARRAY = wdfarray] [, TRANSPOS = transpos] [NOERASEf = noerase]

[, AXROTATE = axrotate] [, AZROTATE = azrotate] [, EDGES = edges]  
 [, ECOLOR = ecolor] [, ELINestyle = elinestyle] [, XRANGE = xrange]  
 [, YRANGE = yrange] [, ZRANGE = zrange] [, XSTYLE = xstyle]  
 [, YSTYLE = ystyle] [, ZSTYLE = zstyle] [, \_EXTRA = extrastuff]

where:

- NORMAL* - An array indicating the normal direction to the plot plane and the position along the normal for the plot plane. The normal is indicated by the flags of 1, 2, or 3 for X, Y, and Z normal. If the number of elements or *NORMAL* is less than *ZVALUE*, the *NORMAL* will be replicated to provide additional values.
- ZVALUE* - An array indicating the value for the normal plane where it intercepts the normal axis. These values are the interval [0,1] (Normalized units)
- STR* - An array of structure array numbers of WDF array numbers to be plotted in each of the normal planes. *STR* must have at least as many values as *ZVALUE* (or 1 value if slice is specified)
- REVERSE* - Plot arrays in reverse order.
- TRANSPOS* - Transpose parameter to go to plotstr. If not specified, *transpos* = 1 for y normal and zero otherwise. If specified with *n* elements then with plot uses *transpos* = *transpos*(i mod n)
- SLICE* - If set, the *k* values and slice coordinates will be set according to the value of normal, *zvalue* and range. **Note:** Range must be set in the direction for slice plotstr, str, normal,  $(zvalue - range(0)) / (range(1) - range(0))$ , ...  
 Default is slice = 0
- WDFARRAY* - If keyword is set, the str values will be interpreted as WDF array numbers. The default is *WDFARRAY* = 0
- NOERASE* - If set the plot will not be erased and the # dimensional transform will be generated. For example, the commands *shade\_surf*, *surface* or *plotstr* with these keywords can be used to set up the transform:  
 plotstr, 1, /save, /shade, & plotcube, /noerase, ..
- AXROTATE* - Rotation about the X axis. Default = 30
- AZROTATE* - Rotation about the Z axis. Default = 30
- EDGES* - An array of integers indicating edges of the reference cube which are to be drawn., Values are:  
 10 or 11 the x axis

- 12 the edge parallel to x axis at y = 1
- 13 the edge parallel to x axis at z = 1
- >13 the edge parallel to x axis at y = 1 and z = 1
- 20 or 22 the y axis
- 21 the edge parallel to y axis at x = 1
- 22 the edge parallel to y axis at z = 1
- >23 the edge parallel to y axis at x = 1 and z = 1
- 30 or 33 the z axis
- 31 the edge parallel to z axis at x = 1
- 32 the edge parallel to z axis at y = 1
- >33 the edge parallel to z axis at x = 1 and y = 1

**ECOLOR**

- Color of line used to plot edges; Default = !p.color. If ecolord has fewer values than edges, color values are recycled.

**ELIENSTYLE**

- Linestyle used to plot edges; Default = 0. If elienstyle values are recycled.

**XYZ RANGE**

- Range of values for the plot

**XYZ STYLE**

- Style values from each axis. Default is style = 1 if range has two values.

**Note:** Any valid additional keywords which can be used with PLOTSTR or PLO can be used.

**Procedure:**

- 1) Set up the 3D transform.
- 2) Plot the edges if requested.
- 3) Repeat over plots
- 4) Rotate to the appropriate normal plane.
- 5) Use plo or plotstr to generate the image

**EXAMPLE:**

Assume lineouts in WDF arrays 1 to 5 corresponding to locations. loc = [1,3,4,5,7]  
Plot these in X-normal planes in reverse order

```
PLOTcube, /reverse, 1, loc/6.0, indgen(5)+1, /wdf
```

Plot the field magnitude on three faces of a cube from structure array 3

```
PLOTcube, [1, 2, 3], [0, 0, 1], 3, /slice, xrange = [0, 5], yrange = [1, 3], $  
zrange = [0, 10]
```

**NOTE:** If all three faces come from the same structure array, only one array must be specified.

For a plot similar to the one above with no labels

```
PLOTCON, a, /over
```

Overplot conductor geometry from structure and larger plots

```
PLOTcube, [1, 2, 3], [0, 0, 1], 3, /slice, xrange = [0, 5], yrange = [1, 3], $
zrange = [0, 10], xticks = 1, yticks = 1, xmargin = [0, 0], ymargin = [0, 0], $
xtickname = [' ', ' '], ytickname = [' ', ' '], title = ''
```

## PLOTFLD

Plot field variable in a contour, surface, or shaded surface plot. This routine will also integrate field values with respect to the x and/or y values, and take lineouts of the values or integrated field values.

**Warning:** The maximum structure array will be used as a working structure array and will contain the two dimensional data used for the contour plot unless the keyword WORK provides a valid working structure number.

```
format: PLOTFLD, A [, FLD] [, KVALUE1] [, VALUE1] [, KVALUE2] [, VALUE2]
(general)
[, BLOCK = block] [,ADD = add] [, MULTIPLY = multiply]
[, KINTEGRATE = kintegrate]
(plot parameters)
[, INDEX = index] [, SMOOTH = smooth] [, FARRAY = farray]
[, NFARRAY = nfarray][, XARRAY = xarray] [, YARRAY = yarray]
[, TITLE = title] [, XTITLE = xtitle] [, YTITLE = ytitle] [, QUIET = quiet]
[, TRANSPLOT = transplot] [, /LEFT] [, /RIGHT]
[, RESULT = result] [, WORK = work]
(plot types)
[, CNTOUR = contour] [, SURFACE = surface][, SHADESURF= shadesurf]
[, XSURFACE = xsurface] [, BXSURFACE = bxsurface]
(contour param)
[, OVERPLOT = overplot] [IGNORE = ignore] [, NLEVEL = nlevel]
[, LEVELS = levels] [, COLOR = color] [, GRID = grid]
[, X RANGE = xrange] [, Y RANGE = yrange] [, CONDUCTOR = conductor]
(integrals)
[, XINTEG = xinteg] [, YINTEG = yinteg] [, START = start]
(lineouts)
[, XLINE = xline] [, YLINE = yline] [, NPOINTS = npoints] [, WIDTH = width]
[,PLINE = pline] [, OUT = out]
(any stuff)
(any contour keywords -- see IDL manual-- examples include:
FILL, X(Y)TYPE, POSITION, X(Y)MARGIN, THICK, FOLLOW, DOWNHILL,
X(Y)TICKLEN, X(Y)STYLE, C_ANNOTATION, C_CHARSIZE, C_LINestyle, ....)
```

**Note:** FILL and DOWNHILL are new feature of IDL

where:

*A* - IDL field structure (a.type = 1) or QS array number  
*FLD* - Field or attribute parameter to be plotted.

**Note:** FLD may be positive or negative; FLD = -(abs(FLD))

**Note:** If FLD is an array, the square root of the sum of the squares will be used.

**Note:** If A has only one attribute (such as an image or a charge density);the this parameter can be omitted.

*KVALUE1,2* - If FLD is a two dimensional field then these values are ignored. If FLD is a three or four dimensional field then these values indicate the coordinates

VALUE1, 2

which are fixed for the two dimensional plots.

- If FLD is a two dimensional field then these values are ignored. If FLD is a three dimensional field then these values indicate the coordinate values for the fixed dimensions in the two dimensional plots. Linear interpolation is used according to the value of VALUE1. If VALUE1 has two elements the average value over the interval is used.

**Note:** If INDEX keyword is set, the units are taken as array indices which range from 1 to the maximum value.

**Note:** If the two elements of VALUE1 span a block boundary, then only the values in the block containing the midpoint will be used. If the midpoint falls on a block boundary, then the result is probably not what you want!!

#### KEYWORD PARAMETERS:

##### General plot keywords

*BLOCK*

- If the field is multi-block, BLOCK can be used to specify one or more blocks to be plotted/integrated/...

*ADD*

- Add a constant or array to the field value

*MULTIPLY*

- Multiply the field by a constant or a vector
- Note:** If F is the field then (multiply\*F + add) will be plotted

*KINTEGRATE*

- If present and not zero, integrate the slice rather than average over the extent of value1(0) to value1(1). The integral will be performed between value1(0) and value1(1)

If kintegrate = 1, a simple integral

If kintegrate = 2, a  $x \, dx$  integral

If kintegrate = 3, a  $x^2 \, dx$  integral

Limitations:

Data must have a single block

Two values of value 1 must exist

*INDEX*

- If present and not zero, then the units of VALUEi will be taken as indices of the field array. Valid values are 1 to the maximum number of values in KVALUEi. Care should be exercised with multiple block data.

*SMOOTH*

- If present and not zero, the two dimensional field will be smoothed using a boxcar filter. The number of cells will be the lowest odd number greater than

or equal to the maximum of [3, SMOOTH]

*FARRAY, NFARRAY* -

For single block data FARRAY is the field quantity being plotted and NFARRAY is ignored. For multiple block data, a uniform grid is generated with NFARRAY by NFARRAY elements. The field quantity is interpolated to this grid.

**Note:** All plots, except contour plots, plot the array, FARRAY. Default value of NFARRAY is 128 if NFARRAY is undefined, or  $\text{abs}(\text{NFARRAY}) > 16$  if NFARRAY is specified. (That is "/nf" would result in a value NFARRAY = 16)

*XARRAY, YARRAY* - Abscissa and ordinate values for the array, FARRAY

*TITLE, XTITLE and YTITLE*-

If these are character strings, they will be used as the plot title and the axis labels. If they are not character strings or are undefined (i.e., delvar, title, xtitle, ytitle) then they will be returned with the values actually used for the plots. The default TITLE is the attribute label and the dataset comment. The default axis labels are the PFF axis labels read with the data.

*TRANSLOT* - If present and not zero, interchange ordinates and abscissa SETDEFAULT can be used to set a default value

*QUIET* - If present and not zero, level information will not be printed. If quiet has a value of 1 (/QUIET or QUIET =1), NO plots will be make.

*LEFT* - Indicates a new plot using the left axis. A single axis will be drawn on the left side of the grid. For this plot XMARGIN will be set to  $\text{XMARGIN} = [\text{!X.margin}(0), \text{!X.margin}(0)]$  and YSTYLE = 8.

**Note:** If LEFT is set, OVERPLOT is set to 0.

Example:

Make a series of lineouts and overplot them on the contour plot. Average the second spatial component between [0,1 ] and take lineouts at .13, .14, .15.

plotstr, 1, 1, 2, [0,1], xl = [0.13,.14,.15], /pl, /left  
plo, 1, 3, /over, /right, /ns, /color, thick = 2

*RIGHT* - Indicates a OVERPLOT making and using the right axis. If RIGHT is present and not zero, then:

1. A single axis will be drawn on the right side of

the existing grid

2. The field data will be overplotted on the existing grid.

Before using this command, a plot should be created using the keyword: /LEFT

**Note:** If RIGHT is set then OVERPLOT and IGNORE are set to 1.

**Note:** After this command all memory of the original y-axis scaling is lost

*RESULT*

- Structure array number to be used to store farray, xarray, yarray, title, xtitle, ytitle. This array contains a single block structure.

*WORK*

- Structure array number to be used as a working array. Default is maxstructure.

#### Plot types

*CNTOUR*

- If present and not zero, a contour plot will be drawn. If no other plot types are selected, a contour plot will be drawn by default.

Note: Keyword is cntour or /cn to avoid problems with color, conductor keywords.

*SURFACE*

- If present and not zero, a surface plot of the field is generated using the interpolated array, FARRAY.

*SHADESURF*

- If present and not zero, a shaded surface plot of the field is generated using the interpolated array, FARRAY.

*XSURFACE*

- If present and not zero, an xsurface plot of the field is generated which allows rotation and resizing of the data using the interpolated array, FARRAY.

*BXSURFACE*

- If present and not zero, an xsurface plot of the field is generated identical to that used for XSURFACE but with a larger window.

#### Contour plot keywords

*OVERPLOT*

- If present and not zero, then the existing grid is to be used. This option is ignored if any type of surface plots are made. With this option set, previous values of KVALUE1 and VALUE1 are used by default as well as the choice of abscissa and ordinate coordinates. This option assumes that the previous plot was from a PLOTFLD, PLOTGRD, PLOTCON, or PLOTPAR command and attempts use the same parameters for the current plot. To use OVERPLOT with an arbitrary grid, set the IGNORE keyword.

- IGNORE* - This keyword is ignored unless OVERPLOT is set. If OVERPLOT is set and IGNORE is set, then all previous plot information is ignored.
- NLEVEL* - Number of contour plot levels. Default is 20, maximum is 90.
- LEVELS* - Array of values for the plot contours. The actual contour levels will be returned in LEVELS if it is a scalar and NLEVEL is not equal to 1.
- C\_LABEL* - Array of integers of value 0 or 1 to indicate if contour lines are to be labeled. Label = 0, no labels, label = 1 all are labeled. Default is no labels on the contour lines. If "/FOLLOW" is used as a keyword C\_label is set to the array [1,0,1,0,1, ...] and every other line is labeled. If C\_LABEL= make\_array(30, /fix, val= 1), then every contour is labeled.
- COLOR* - Array of colors for the contour lines. If color is not an array, then:  
 If color = -1 then colors will go from 9 to !d.n\_colors uniformly spaced using the existing color table  
 If color = -2 then the 7 primary colors will be used to make a color map. With 20 levels we will have 3 red, 3 green, 3 blue, 3 purple, 3 orange, 3 light blue, and 2 yellow.  
 If color = -3 the 7 standard colors will be cycled.  
 If color is positive, that color will be used for all contours, [1, 7] are the primary colors, 9 to !d.n\_colors depend on the color map.  
 Color = 0 is the default plot color ie white or black  
 Default is set by SETDEFAULT or QSINIT  
 The colors are: red(1), green(2), blue(3), purple(4), orange(5), light blue(6), yellow(7).
- XRANGE, YRANGE* - Two element arrays indicating the desired axis scaling. If they are scalar or undefined, the actual data range will be returned in these variables. If only one is specified, the other will be set based on the limited range or domain of the data.
- CONDUCTOR* - Specifies a conductor structure or structure array number specifying a conductor structure. If valid, the conductors will be overplotted. Colors and thicknesses can be set with SETDEFAULT.
- GRID* - Specifies a grid structure or a structure array number. If valid, the grid will be overplotted. Color and thickness can be set with SETDEFAULT.

## INTEGRAL keywords

*XINTEG*

- Integrate the field variable with respect to the abscissa value (x) according to the following values:

If  $XINTEG = 0$ , do nothing.

If  $XINTEG = 1$ , integrate  $f(x, y) dx$

If  $XINTEG = 2$ , integrate  $f(x, y) x dx$

If  $XINTEG = 3$ , integrate  $f(x, y) x^2 dx$

where  $dx$  is the spacing between abscissa values. The initial value of the integral is set to 0.0 or to the value in the WDF array specified by *START*.

**Note:** If two blocks terminate on a third, then the initial value of the integral at the block boundary is linearly interpolated between the two incoming blocks.

*YINTEG*

- Integrate the field variable with respect to ordinate value (y) according to the following values:

If  $YINTEG = 0$  or if  $XINTEG = [1,2,3]$ , do nothing.

If  $YINTEG = 1$ , integrate  $f(x, y) dy$

If  $YINTEG = 2$ , integrate  $f(x, y) y dy$

If  $YINTEG = 3$ , integrate  $f(x, y) y^2 dy$

where  $dy$  is the spacing between ordinate values. The initial value of the integral is set to 0.0 or to the value in the WDF array specified by *START*.

**Note:** If  $XINTEG \neq 0$  then *YINTEG* is ignored.

**Note:** If two blocks terminate on a third, then the initial value of the integral at the block boundary is linearly interpolated between the two incoming blocks.

*START*

- If present this is interpreted as initial values at the lower limit of integration. *Start* is an integer indicating a WDF array. Values are between 1 and *maxwdfarray*. Any blocks with coordinates in the domain of *START* will be initialized unless they are connected directly to another block. Values at one point on a boundary are propagated along the entire boundary and linearly interpolated between adjacent values.

## LINEOUT keywords

**Note:** Values of the *LINEOUT* coordinates are determined in the following sequence.

1. Two elements in width - These values taken as the max/min value
2. *NPOINTS* >0, *NPOINTS* equally spaced lineouts in specified direction(s), spacing is between the specified range -- use *xrange* and *yrange* to limit the

values of the lineouts.

### 3. XLINE, YLINE is an array of values for the lineout

**Note:** To take a single lineout at one location width must be specified with two values (which may be equal) and /xline or /yline specified to indicate the direction of the lineout.

**or** NPOINTS = 1 then one lineout will be taken at the appropriate y or x location specified by XLINE or YLINE with a width specified by WIDTH

**Note:** If the width of a lineout spans a block boundary, only values in the block containing the midpoint of the lineout will be used.

- |                |  |
|----------------|--|
| <i>XLINE</i>   | - If present and not zero, a WDF array of field value vs. the abscissa is generated at designated ordinate values.   |
| <i>YLINE</i>   | - If present and not zero, a WDF array of field value vs. the ordinate is generated at designated abscissa values.   |
| <i>NPOINTS</i> | - Indicates number of lineouts in the X and/or Y direction.  |
| <i>WIDTH</i>   | - If a single value, then it is taken as the width of the lineout. Default value is 0. If two values are present, they are taken as the minimum and maximum values for obtaining the lineout. In all cases, the average value between the limits or in the Width about the designated point is used for the lineout. |

**Note:** If npoints = 1, xline and/or yline are assumed to equal the desired value of the lineout.

- |              |   |
|--------------|---|
| <i>PLINE</i> | - If present and not zero, plot a dashed line at the position of the lineouts                                   |
| <i>OUT</i>   | - Value of the first WDF array for storing lineouts or an array of values for the lineouts.<br>Default is OUT=1 |

**Note:** An alphabetical listing of the keywords:

add, block, bxsurface, cntour, color, conductor, farray, grid, ignore, index, kintegrate, levels, multiply, nfarray, nlevel, npoints, out, overplot, pline, quiet, result, smooth, shadesurf, start, surface, title, transplot, width, work, xarray, xinteg, xline, xrange, xsurface, xtitle, yarray, yinteg, yline, yrange, ytitle

**EXAMPLE:**

Contour plot of the charge density, in structure F, from a 2-D calculation.

**PLTFLD, f**

Contour plot of the charge density, in structure F, from a 3-D calculation taking a slice for the second dimension equal to 0.0

**PLOTFLD, f,1, 2, 0.0**

or

**PLOTFLD, f, 2, 0.0 (since only one attribute, 1 can be omitted)**

Plot a slice of an image file, and take lineouts in the x and y directions through a feature in the contour plot. Assume the image is in structure array 4 and an average over k components 3 to 10 is desired. The lineout width is 0.01 and lines indicating lineouts are to be plotted.

**Note:** Plotstr and plotfld may be used interchangeably.

**PLOTSTR, 4, 3, [3, 10]**

**cur, x, y (place cursor over the center of the feature)**

**PLOTSTR, 4, 3, [3, 10], np=1, xline = y, yline = x, wid = 0.01, /pl**

Contour plot of the CR-39 track distribution with the third dimension the track size index in structure, C. Average tracks in bins 2 through 10.

**PLOTFLD, c, 3, [2, 10], /index**

or

**PLOTFLD,1, c, 3, [2, 10], /index**

**Note:** Add ", m = 9" to plot the total number of tracks (since average is over 9 bins (2 to 10))

Contour plot of the magnitude of the vectors in structure START from TWOQUICK simulation (ie. Only two spatial dimensions).

**PLOTFLD, start, [1,2,3]**

Contour plot of two dimensional structure array 1 of charge density (i.e. only 1 attribute)

**PLOTFLD, 1**

Contour plot of vector 1 of field structure a. Use a value of 0.4 for spatial dimension 2 and take lineouts at 10 locations in the first and third dimensions and store the lineouts in WDF arrays 11 to 30. Sequence colors over the contour lines.

**PLOTFLD, a, 1, 2, 0.4, /xl, /yl, npoint = 10, out = 11, color=-1**

Plot from structure a, vector 3 in a contour plot after smoothing. Set the value of the first dimension to 0.05 and sequence linestyles over the contours. Take a lineout between values of the second dimension of 1 and 2 and store in WDF array 1.

**(Note:** The FOLLOW keyword should be used with multiple line types.)

**PLOTFLD, a, 3,1,0.05,/smo,c\_line = findgen(6),/follow, /yl, wid=[1,2]**

or

**PLOTFLD, a, 3,1,0.05,/smo,c\_line = findgen(6),/follow, yl=1.5, wid=1, np =1**

Make a shaded surface plot from structure END, vector 1, after smoothing, and generate vertical lineouts at intervals of 0.1 between 1 and 2.5. Assume

structure END has two spatial components.

PLOTFLD, end,1, yline= findgen(16)\*0.1+1, /smo, /sha

## PLOTGRD

Plot computational grids or block boundaries.

format: PLOTGRD, A [, X] [, Y] [, ZVALUE] [, YRANGE = YRANGE] [, THICK = THICK]  
 [, XRANGE = XRANGE] [, COLOR = COLOR] [, TITLE = TITLE]  
 [, XTITLE = XTITLE] [, YTITLE = YTITLE] [, CHANGESIGN = CHANGE]  
 [, W1= WINDOW1] [, W2= WINDOW2] [, W3= WINDOW3] [, SKIP= SKIP]  
 [, XSTYLE = XSTYLE] [, YSTYLE = YSTYLE] [, XTYPE = XTYPE]  
 [, YTYPE = YTYPE] [, LINSTYLE = LINSTYLE] [, BLOCK = BLOCK]  
 [, XARRAY = XARRAY] [, YARRAY = YARRAY] [, WARRAY = WARRAY]  
 [, CONDUCTOR = CONDUCTOR] [, PRINTFILE = PRINTFILE]

where

- A - IDL grid structure (type = 3)
- X - Abscissa value (1, 2, or 3 depending on desired spatial array)
- Y - Ordinate value (1, 2, or 3 depending on desired spatial array)
- Zvalue - For three dimensional data, value of plane in the third dimension. Default is them mean of the minimum and maximum.

**Note:** In the case of an overplot, the x and y values will be determined from the previous plot.

### KEYWORD PARAMETERS:

- OVERPLOT* - If not zero, indicates no new grid is to be drawn.
- THICK* - Indicates line thickness for the grid default is 0.6
- LINSTYLE* - Line type for the grid. An array may be specified to sequence over line styles. Lines specified by LINSTYLE are: solid(0), dot(1), dash(2), dot-dash(3), dot-dot-dot-dash(4), and long dash(5).
- BLOCK* - The block of data to be plotted. Default is all blocks
- COLOR* - Indicates the color of the grid, Color may be an array to sequence over colors. If colors is an array, the minimum value is 1 and the maximum is !d.n\_colors. The colors are: black(0), red(1), green(2), blue(3), purple(4), orange(5), light blue(6), yellow(7) and white(8).
- SKIP* - The number of grid lines plotted is reduced by this value.
- CONDUCTOR* - **Note:** If skip is very large, the blocks will be outlined.  
 - Specifies a conductor structure or a structure array number specifying a conductor structure. If valid, the conductors will be overplotted. Colors and thicknesses can be set with SETDEFAULT.

*TITLE, XTITLE and YTITLE*

- If grid is not zero, these will be used for axis and the plot labels. If these are not string arrays, the title, xtitle and ytitle used for the plots will be used.
- XARRAY, YARRAY* - are the grid arrays for the first block plotted (X,Y)RANGE, STYLE, TYPE have the standard IDL meaning.
- PRINTFILE* - Specifies a print file for printing the x and y grid lines in the plot. If PRINTFILE is not a sting, the information will be sent to the screen.

**EXAMPLE:**

Plot from structure g, space 1 vs. space 3 with blue dashed lines

```
PLOTGRD, g, 3, 1, COLOR = 3, LINESYLE = 2
```

Plot from structure g, space 1 vs. space 3 with the first line green and the next 4 red. Print the values of the grid lines to the terminal.

```
PLOTGRD, g, 3,1, COLOR = [2,1,1,1,1], /print
```

**PLOTPAR**

Plot particle arrays with color weighting

**format:** PLOTPAR, A [, X] [, Y] [, WT] [, YRANGE = YRANGE] [, THICK = THICK] [, XRANGE = XRANGE] [, COLOR = COLOR] [, TITLE = TITLE] [, XTITLE = XTITLE] [, YTITLE = YTITLE] [, CHANGESIGN = CHANGE] [, FOLLOW = follow] [, SYMSIZE = SYMSIZE] [, FAST = FAST] [, W1= WINDOW1] [, W2= WINDOW2] [, W3= WINDOW3] [, SKIP= SKIP] [, XARRAY = XARRAY] [, YARRAY = YARRAY] [, FARRAY = FARRAY] [, OVERPLOT = OVERPLOT] [, CONDUCTOR = CONDUCTOR] [, GRID = GRID] [, RESULT = result] [, QUIET = quiet] [, LEFT = left] [, RIGHT = right] [\_EXTRA = extrastuff]

where:

- A* - IDL particle structure (type = 2)
- X* - Abscissa values (positive = x array, negative = attribute array)
- Y* - Ordinate values (positive = x array, negative = attribute array)
- WT* - Weight array (positive = x array, negative = attribute array)

**Note:** If x, y, or wt is an array, then the square root of the sum of the squares will be used.

**Note:** If overplot is not zero, then x and y will be determined from the previous plot. WT will be the second parameter.

- PSYMBOL* - Symbol type - Default is 8
- FAST* - Indicates a fast plot is desired. If PSYMBOL is not specified and fast is not zero, a triangle will be used.

- OVERPLOT* - If not zero, indicates no new grid is to be drawn.
- SYMSIZE* - Symbol size multiplier - Default is 0.5
- COLOR* - If wt has fewer elements than the x and y arrays, then the points will be drawn using color = color. The colors are: black(0), red(1), green(2), blue(3), purple(4), orange(5), light blue(6), yellow(7) and white(8).
- CHANGESIGN* - If change is present and not zero, wt = -wt
- SKIP* - The total number of points plotted is reduced by this value.
- W1, W2, W3* - Each is an array of 3 or more elements. Plot can be windowed in real space or attribute space. Last two values in the array are the minimum and maximum values of the window parameter; the first value(s) are the spatial or attribute parameters to be used for the window. (window parameter is positive = spatial array, negative = attribute array) (in the case of multiple parameters, use the sq root of the sum of the squares)
- CONDUCTOR* - Specifies a conductor structure or a structure array number specifying a conductor structure. If valid, the conductors will be overplotted. Colors and thicknesses can be set with setdefault.
- FOLLOW* - If a particle dataset represents the motion of a test particle, the follow command allows the orbit in one plane to be mapped.
- GRID* - Specifies a grid structure or a structure array number. If valid, the grid will be overplotted. Color and thickness can be set with setdefault.
- TITLE, XTITLE and YTITLE* - If over is zero, these will be used for axis and the plot labels. If these are not string arrays, the comment label, xlabel and ylabel in the structure will be used.
- XARRAY, YARRAY, FARRAY* - are the actual arrays sent to PLOTWT for the particle plots unless BIN ne 0. See BIN below.
- BIN* - If present and not zero, then XARRAY, YARRAY and FARRAY become X and Y coordinates for a contour plot of the particle density and the actual function. ie: "contour, farray, xarray, yarray" would produce a contour plot of the particle density. If WT is the particle charge, then FARRAY is the

- charge density. If WT is absent, then FARRAY is the number density. The range of xarray and yarray are the actual plot ranges for the particle plots.  
 If BIN is less than 16 then a 64x64 grid will be established for FARRAY.  
 If BIN is 16 or larger then a BINxBIN grid will be established for FARRAY.
- RESULT* - If present and a valid structure array number, the xarray, yarray, and farray data from the bin command will be sent to this structure array.
- QUIET* - If present and not zero, no plots will be made.  
**Note:** If quiet is used the axis scaling for the BIN option will use the exact range of X and Y values unless xrange and yrange are specified.
- LEFT* - Indicates a new plot using the left axis. A single axis will be drawn on the left side of the grid. For this plot XMARGIN will be set to XMARGIN = [!X.margin(0), !X.margin(0)] and YSTYLE = 8.  
**Note:** If LEFT is set, OVERPLOT is set to 0.
- RIGHT* - Indicates a OVERPLOT making and using the right axis. If RIGHT is present and not zero, then:  
 1. A single axis will be drawn on the right side of the existing grid  
 2. The particle data will be overplotted on the existing grid.  
 Before using this command, a plot should be created using the keyword: /LEFT  
**Note:** If RIGHT is set then OVERPLOT is set to 1.  
**Note:** After this command all memory of the original y-axis scaling is lost.
- \_EXTRA* - These parameters are sent to PLOTWT and can have any value appropriate for the plot command including the following:  
 (X,Y)RANGE, (X,Y)STYLE, (X,Y)TYPE, LINESTYLE, POSITION, etc and have the standard IDL meaning.

**EXAMPLE:**

Plot from structure a, space 1 vs. space 3 with attribute 4 as the weight and use a grid [0,0.064, 0, 0.14]

```
PLOT PAR, a, 3, 1, -4, xrange = [0,0.064],yrange = [ 0, 0.14]
```

Plot from structure a, the sum of the squares of attributes 1 and 3 vs. space 3 with attribute 4 as the weight

```
PLOT PAR, a, 3, [-1,-3], -4
```

Plot from structure a, space 1 vs. space 3 and limit the plot to values of space 2 in the range [0, 3] with attribute 4 as the weight and overplot conductors in datasets 3 and 4

**PLOT**PAR, a, 3, 1, -4, w1 =[3, 0, 3], cond= [3,4]

Overplot a particle plot, with attribute 4 as a weight and the conductors in structure, c2.

**PLOT**PAR, a, -4, /over, con = c2

Plot from structure a, space 1 vs. space 3 with a window of energy of 1e10 to 1e13 for the magnitude of the momentum in attributes [1,2,3] with attribute no weight

**PLOT**PAR, a, 3, 1, w1 =[-1, -2, -3, 1e10, 1e13]

## PLOTSTR

Plot a structure array. This is a simple command to plot all structure types.

**Warning:** The maximum structure array will be used as a working structure array for PLOTFLD

For a description of parameters and keywords, see the commands

PLOTFLD  
PLOTCON  
PLOTGRD  
PLOTPAR

Example:

Contour plot of the charge density, in structure F, from a 2-D calculation.

**PLOT**STR, f

Contour plot of the charge density, in structure array 2, from a 3-D calculation taking a slice for the second dimension equal to 0.0

**PLOT**STR, 2,1, 2, 0.0

or

**PLOT**STR, 2, 2, 0.0 (since only one attribute, 1 can be omitted)

Plot from structure g, space 1 vs. space 3 with blue dashed lines

**PLOT**STR, g, 3, 1, color = 3, linestyle = 2

Plot from structure array 2, space 1 vs. space 3; use a grid [0,0.064, 0, 0.14]

**PLOT**STR, 2, 3, 1, xrange = [0,0.064],yrange = [ 0, 0.14]

## PLOTVEC

Make a vector plot from two single block field structures or from two attributes in a single block field structure. This routine is based on VELOVECT. Two options exist for the vector:

1. A spot with a bar going in the direction of the vector.

2. An arrow going in the direction of the vector.

If the length parameter is negative, all arrows/bars will be the same length. If color is -1, then the arrows/bars & spots will be colored according to their length and the existing color map.

**format:** PLOTVEC, STR1 [, STR2 or VECTORS] [, XRANGE = xrange]  
 [, YRANGE = yrange] [, ARROW = arrow] [, LENGTH = length]  
 [, BIN = bin] [THICK = thick] [, SYMSIZE = symsize] [COLOR = color]  
 [, BORDER = border] [, XBORDER = xborder] [, YBORDER = yborder]  
 [, XSTYLE = xstyle] [, YSTYLE = ystyle] [, TITLE = title]  
 [, TRANSPLOT = transplot] [, OVERPLOT = overplot]  
 [, \_EXTRA = extrastuff]

where:

- STR1* - array or array number of the first field structure array - this must be two dimensional or three dimensional with a degenerate dimension and a single block.  
 If STR2 is given then STR1 must be a scalar field.  
 If STR2 is not given then STR1 must have 2 or more vector components.
- STR2* - array or array number of field structure to be plotted with the first structure. The field in this structure represents the y-component of the vector.  
 - this must be two dimensional or three dimensional with a degenerate dimension, a single block, and have a scalar field.

or

- vectors* - array of two elements indicating the vector components to be used for the X and Y components of the vector plot. Default value is the components corresponding to the nondegenerate dimensions.
- xrange* - Range of X-values of interest for the plot
- yrange* - Range of Y-values of interest for the plot
- arrow* - If present and not zero, vectors will be indicated with an arrow. (based on USERLIB routine, VELOVECT) Default is ARROW =0; vectors indicated with a spot and bar.
- length* - Length factor in units of the point spacing (see BIN)  
 If length is negative, then all bars/arrows will be the same length. Magnitude can be shown with color. Default is the length of a cell = 1.
- bin* - Number of points in the horizontal and verticle direction. If BIN is a vector then the [BIN(0), BIN(1)] are the number in the horizontal and vericle direction. If BIN is a scalar, Then that number will

- be used for both directions.  
Default is BIN = [30, 30].
- thick* - Thickness of the arrow or bar used for the vector.  
Default is 2
- symsize* - Size of the spot used for the vector origin if  
ARROW = 0. Default is 0.75
- color* - Array of colors for the contour lines. If color is not  
an array, then If color = -1 then colors will go from  
9 to !d.n\_colors uniformly spaced using the existing  
color table If color is positive, that color will be used  
for all contours, [1, 7] are the primary colors, 9 to  
!d.n\_colors depend on the color map. Color = 0 is  
the default plot color (ie, white or black) which is  
set by the system parameter, !P.COLOR. Default is  
set by SETDEFAULT or QSINIT. The colors are:  
red(1), green(2), blue(3), purple(4), orange(5), light  
blue(6), yellow(7).
- title* - Title for the plot; Default is a combination of the  
attribute or vector labels.
- transplot* - If present and not zero, interchange ordinates and  
abscissa. SETDEFAULT can be used to set a default  
value  
**Note:** This only switches spatial coordinates. To  
switch vector coordinates, interchange the two  
vector components in *VECTORS* or the two  
structures, *STR1* and *STR2*.
- overplot* - If present and not zero, plot the vector plot on the  
current plot
- border* - If present and not zero, add a border to X RANGE  
and Y RANGE given by BORDER\*LENGTH\*point  
spacing. XSTYLE and YSTYLE, if not set, are set to  
1 to force exact axis scaling.
- x(y)border* - Ignored if border is set. This is the same as border  
but it only applies to the X (or Y) axis
- x(y)style* - Have the standard IDL meaning for the plot  
command.
- extrastuff* - Any other parameters which can be used with the  
plot command. This keyword is not used if *overplot*  
is present and not zero.

**Examples:**

Structure array 2 has a degenerate middle dimension and 3 components of the electric field. Make a vector plot of the first and third components of the electric field and make a plot of the first and second components of the field.

**PLOTVEC, 2 ; (Components default to [3,1])**

**PLOTVEC, 2, [2,1]**

Structure 3 is a 3D, multiblock electric field structure. In the plane  $z = 0.5$ , plot the x and y components of the electric in a vector plot with arrows. Use structures 5 and 6 as working structures.

**Note:** If only a limited portion is of interest for the vector plot, use *xrange*, *yrange*, or *block* keywords to limit the region in the result structures.

**PLOTSTR, 3, 1, 3, 0.5, result =5**

**PLOTSTR, 3, 2, 3, 0.5, result = 6**

**PLOTVEC, 5, 6, /arrow**

To overlay a vector plot on a contour plot for the above example.

**PLOTSTR, 3, [1,2], 3, 0.5**

**PLOTVEC, 5, 6, /arrow, color = 4, /over, /ignore, len = -0.75**

Structure array 2 has the radial component of the electric field and structure array 5 has the azial component of the electric field; the structures are degenerate in the theta direction. Make a vector plot of these field values.

**PLOTVEC, 5, 2**

## PLOTWT

Plot two arrays with color weighting.

**Note:** This utility is called by PLOTPAR. No clipping is performed in order to speed the plotting. If *XRANGE* or *YRANGE* are specified and do not include the full data range, then points will be plotted exterior to the grid.

**format:** PLOTWT, x, y, wt, [, PSYMBOL = *psym*] [SYMSIZE = *symsize*]  
 [, OVERPLOT = *overplot*] [, COLOR = *color*] [, TITLE = *title*]  
 [, XRANGE = *xrange*] [, YRANGE = *yrange*] [YSTYLE = *ystyle*]  
 [, XSTYLE = *xstyle*] [, XTYPE = *xtype*] [, FAST = *fast*]  
 [, XTITLE = *xtitle*] [, YTITLE = *ytittle*] [, STUFF = *extrastuff*]

where:

- x* - Abscissa values
- y* - Ordinate values
- wt* - Weight array used to determine color or the points  
 Default is  $WT = 1$ , i. e. no weighting  
 Note: If electron charge is used, use *-wt*.
- psym* - Symbol type - Default is 8 or 5 (see FAST)
- fast* - Indicates a fast plot is desired. If PSYMBOL is not specified and fast is not zero, a triangle will be used.
- symsize* - Symbol size multiplier - Default is 0.3
- overplot* - If not zero, indicates no new grid is to be drawn.
- color* - If wt has fewer elements than the x and y arrays,

then the points will be drawn using `color = color`. The colors are: black(0), red(1), green(2), blue(3), purple(4), orange(5), light blue(6), yellow(7) and white(8).

- xrange, yrange* - two element arrays with the first element equal to the minimum and the second element, the maximum value of the x-axis and y-axis respectively. These values will be rounded to "nice" values unless *xstyle* and/or *ystyle* have been set.
- xstyle, ystyle* - integers to specify the axis style. Setting bit 0 (a value of 1 or /XS (/YS)) forces exact axis range. Bit 1 (value 2) extends the axis range. Bit 2 (value 4) suppresses the entire axis. Bit 3 (value 8) suppresses a box style axis. Bit 4 (value 16) inhibits setting the Y axis minimum value to zero (Y axis only).
- xtitle, ytitle, title* - strings to be used for the X and Y axis labels and the plot title.
- xtype, ytype* - specify logarithmic axis if present and nonzero
- extrastuff* - Structure sent to the plot command using the `_EXTRA` keyword

Examples:

Plot Y versus X with charge used as a weight

`PLOTWT, x, y, charge`

## PRESET

Reset font, line thickness, and multi plots.

**format:** `PRESET`

Examples:

Reset the default plot parameters

`PRESET`

## PRIAR

Prints the parameters of a WDF array. Maximum and minimum values, the number of points, the file from which it was read, and the character strings associated with each of the labels is printed with ordinate values.

**format:** `PRIAR, wd1 [, NPOINT = npoint]`

where:

- wd1* - WDF array number
- NPOINT* - specifies the number of ordinate values to be printed. If *npoint* is negative, all values will be

printed. If npoint is omitted, up to 72 values will be printed.

Examples:

Print the parameters and up to 72 values of array3

```
PRIAR, 3
```

Print the parameters and the entire set of ordinates for array 4

```
PRIAR, 4, np =-1
```

## PS2X

Switch from a postscript file to X windows for plotting. The old previous X fonts and thickness parameters are reset.

format: **PS2X**

## PUT\_ARRAY

This procedure puts an array into a structure the dimensions of the array must be identical to the existing vector which it replaces

format: **PUT\_ARRAY, structure, vector, space [, block] [, lab= lab]**

where:

- |                  |   |
|------------------|---|
| <i>structure</i> | - IDL structure name or structure array number type must be<br>3. field<br>4. particle<br>5. grid       |
| <i>vector</i>    | - vector to be inserted into structure. It must be identical to the vector it replaces.                 |
| <i>space</i>     | - if not zero indicates spatial dimension desired, if negative indicates field data and attribute data. |
| <i>block</i>     | - indicates block - not used for particle data  |
| <i>label</i>     | - new label to be used for the attribute or spatial dimension   |

Examples:

Use IDL internal functions to operate on structure data

```
im = get_array(1, -3)
```

```
im = sin(im)
```

```
PUT_ARRAY, 1, im, -3
```

## PUTX

Store an array of abscissas (x-values) into a WDF array.

**Note:** PUTX assumes that the array is uniform and uses only the first and last values to calculate the correct interval for the number of points in the WDF array.

**format:** PUTX, *xarray*, *wd1*

where:

- xarray* - array used to calculate an initial point and a spacing for a WDF array
- wd1* - WDF array number

Example:

Put the IDL array *time* into WDF array 4

**PUTX, time, 4**

## PUTY

Store an array of ordinates (y-values) into a WDF array.

**Note:** The array must have the same number of elements as the original array.

**Note:** PUTY can be used with GETY to perform IDL functions on an array.

**format:** PUTY, *yarray*, *wd1*

where:

- yarray* - array to be stored as ordinates in a WDF array
- wd1* - WDF array number

Examples:

Put the IDL array *voltage* into WDF array 4

**PUTY, voltage, 4**

Calculate the square root of the contents of array 4

**PUTY, sqrt ( getyf(4) ), 4**

## PWD

Show current directory

**format:** PWD [, CURRENT = *current*]

where:

- CURRENT* - Name of current directory

Examples:

Show current directory

**PWD**

## RE

Reads a PFF dataset into a WDF array. Procedure options are explained in Table 1.

**Note:** If *wda* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then the command will be aborted.

**format:** RE, *wda* [, *dataset*] [, NARRAY = *narray*] [, ALL = *all*]

[, BLOCK = block] [, FILEID = fileid]

where:

- wda* - WDF array number or an array of numbers
- dataset* - indicates a PFF dataset number, an array of dataset numbers, or is a search string for dataset comments. If *dataset* is missing or zero, then it is set to the current dataset.  
**Note:** If *dataset* is a string, a leading “^” indicates that the string begins at the start of the dataset comment.
- all* - nonzero value indicates all datasets with the search string are to be read into WDF arrays. If *all* is missing or zero, only the elements of *wda* will be filled.
- fileid* - file id of the PFF file containing the data
- narray* - number of datasets read into WDF arrays
- block* - block of data in the dataset  
 default is 1

**Table 1: Procedure options for reading PFF datasets into WDF arrays**

<i>wda</i>	<i>dataset</i>	Action
array(N)	zero or missing	Read current dataset and (N-1) successive datasets into the array(N) WDF arrays.
integer	array(M)	Read array(M) datasets and store in WDF array <i>integer</i> and the following (M-1) WDF arrays.
array(N)	array(M)	M must equal N; read the array(M) datasets and store in the array(N) WDF arrays.
integer	string (all = 0)	Read dataset with <i>string</i> into WDF array, <i>integer</i> If more than one array with <i>string</i> , look for <i>string</i> followed by spaces at the beginning of the dataset comment.
integer	string (all ≠ 0)	Read all datasets with <i>string</i> into WDF array, <i>integer</i> and successive WDF arrays.
array(N)	string	Read all datasets with <i>string</i> into WDF arrays, array(N). Number of array elements must equal number of datasets with <i>string</i> in the dataset comment. If <i>string</i> occurs in only 1 dataset, then read that dataset and the next (N-1) datasets into WDF arrays, array(N).

Examples:

Read dataset 34 into array 1 from file with file id of 4.

RE, 1, 34, fil = 4

Read into WDF array 2,..., 10, datasets 34,..., 42 from the current file

RE, indgen(9)+2, 34 ; or

RE, 2, indgen(9)+34

Read into WDF arrays beginning with 2 all datasets beginning with "mspin" in the current file and indicate the number of arrays in the variable, *nread*.

RE, 2, '^mspin', n=nread, /all

Read into WDF arrays [2, 5, 8, 11] the datasets [3, 4, 1, 2] from file with id of 2

RE, [2,5,8,11], [3,4,1,2], file= 2

Read into WDF arrays [2,5,8, 11] the datasets having the string "vtot".

**Note:**"vtot" must occur in either 1 array or 4 arrays.

Re,[2,5,8,11], 'vtot'

Read into WDF array *volt* the dataset having the string "vctot"

**Note:** *Volt* may be an undefined variable name; it will be set equal to the lowest unused WDF array.

RE, volt, 'vctot'

## READ\_PE

Read a Perkin Elmer formatted dataset into IDL

format: READ\_PE , File\_Name, image, space [, TLABEL = tlabel]  
 [, CLABEL = clabel] [, BLABEL = blabel] [, XLABEL = xlabel]  
 [, YLABEL = ylabel] [, HEADER = header] [, NPOINTS = npoints]  
 [, DELTA = delta] [, OUTFILE = outfile] [, FILTER = filter] [, SCALE = scale]  
 [ SWAP = swap][, PATH = path] [, XIMAGE= ximage]  
 [XHEADER = xheader] [, PHEADER = pheader]

where:

- |                  |  |
|------------------|--|
| <i>File_Name</i> | - the name of the Perkin Elmer file with or without an extension . <b>Note:</b> Any extensions will be removed and extensions of .hdr and .img will be assumed. (See XHEADER and XIMAGE below) |
| <i>image</i>     | - array containing the image file  |
| <i>space</i>     | - array containing the dataset x and y arrays. Use SP2XYZ, image, space, x,y to obtain x and y (SPACE = [X, Y])<br><b>Note:</b> MicroD offsets are ignored. Both X and Y start at (0, 0)       |
| <i>TLABEL</i>    | - Dataset type label. Default is the value of "SETVAL" (Specular Density, Transimission of Intensity)  |

<i>CLABEL</i>	- Dataset comment label. Default is File Name // Comment1 // Comment2 // ScanTime
<i>BLABEL</i>	- title or label for this block. Default is Specular Density, Transmission or Intensity depending on the value of "SETVAL"
<i>XLABEL</i>	- horizontal dimension label for this block. Default is 'X (microns)'
<i>YLABEL</i>	- Vertical dimension label for this block. Default is 'Y (microns)'
<i>HEADER</i>	- Complete header for the PE dataset
<i>NPOINTS</i>	- Array of points for each dimension
<i>DELTA</i>	- Array of microns/step for each dimension
<i>OUTFILE</i>	- File name without path information
<i>FILTER</i>	- Optional filter. If specified this parameter must have at least two values which are taken as the maximum and minimum of the scanned data. If three parameters are given, the third is taken as the default value given to all data which falls outside the specified range. Default value is [0, 5000, -1] . That is all data less than 0 and greater than 5000 will be set to -1. With the default scaling of 800, this corresponds to values between 0 and 6.25
<i>SCALE</i>	- Parameter used to scale the integer counts. Default is $1/800 = 1.25e-3$
<i>SWAP</i>	- This keyword is set by default, ie swap = 1. If swap = 0, then the bytes in image will not be swapped. <b>Note:</b> VAX data must be swapped.
<i>PATH</i>	- Directory path to the desired file. This is the same default value as used in openpff
<i>XIMAGE</i>	- Extension for the image file. Default is .Img
<i>XHEADER</i>	- Extension for the header file. Default is .Hdr
<i>PHEADER</i>	- If the keyword is set, the header will be printed to the screen.

#### Examples:

Read the file S6424n4f and retrieve all the parameters for writing the file as a pff file. Print the header.

```
READ_PE, 'S6424n4f', image, space, tlab = tlab, clab = clab, blab = blab, xlab = xlab, ylab = ylab, /ph
```

**Note:** to write the above data first insure a PFF file is open or if necessary create the file and then write it:

```
creatpff, 'S6424n4f.pff'
```

```
writepff, 0, imagem space, tlab = tlab, clab = clab, blab = blab, xlab = xlab, ylab =
```

y<sub>lab</sub>

## READASC

Read a text file into an array or two arrays. This procedure assumes that each record in the file has either a single value or a single abscissa value followed by one or more ordinate values. Several formats are supported. If the filename is specified, then the file will be opened and closed by the procedure. If a unit is specified, then data is read from the current position in the file.

**Note:** Present version is limited to 4096 records unless the number of points is specified in the file or on the command line. The procedure attempts to read data until the end-of-file is reached.

**format:** READASC, filename, nvar, x, y [, format = format] [, np = np] [, label = label]

where:

- |                 |   |
|-----------------|---|
| <i>filename</i> | - name of the file containing the data<br>If filename is 0, then pickfile will be used to select a file. This file is opened, data is read, and the file is closed after reading data.<br>If filename is a number, then no file is opened but the procedure reads from unit = filename. |
| <i>nvar</i>     | - number of values in each record   |
| <i>x</i>        | - first value in each record<br><i>x</i> is dimensioned $x(np)$   |
| <i>y</i>        | - all additional values in the record<br><i>y</i> is dimensioned $y(np, nvar-1)$  |
| <i>np</i>       | - number of data records in the file  |
| <i>label</i>    | - string array of labels of the nvar-1 arrays   |
| <i>format</i>   | - integer indicating the data format<br>0 Default - only data values stored in the file<br>1 number of data points followed by the data values<br>2 (nvar-1) labels (stored in <i>label</i> ) followed by the data<br>3 number of points, followed by labels and the data.              |

Example:

Read a text file with 4 values per record into arrays: *time* and *velocity*. Separate velocity into its components

```
READASC, 'users/temp/accel.dat', 4, time, velocity
```

```
vx = velocity(*, 1) & vy = velocity(*, 2) & vz = velocity(*, 3)
```

Read a text file with the number of points, np, followed by np X-Y pairs. Pick the name of the file using PICKFILE and store the data in x and y.

```
READASC, 0, 2, x, y, form= 1
```

Read from unit = 2 two arrays of data and store in s and t.

**READASC, 2, 2, s, t**

Read a text file, *file\_name.dat*, with 3 values per record into arrays, *x* and *y*. The first 4 lines of the file are:

```
512
Current
Voltage
0.0000E+00 0.0000E+00 0.0000E+00
.....
```

Store the data in WDF arrays 3 and 4 using the labels as the dataset comment. An error will occur if the file contains more than 515 records.

```
READASC, 'File_name.dat', 3, x, y, label= label, format = 3
```

```
for i = 0, 1 do i2w, x, y(*, i), 3+i, c = label(i)
```

or (using a different command)

```
READASC , 'File_name.dat', 2, 3
```

## READCON

Reads a CONDUCTOR dataset into an IDL structure.

Procedure options are explained below. PFF dataset type must be VTX

**format:** READCON, structure [, dataset] [, number] [, FILEID = fileid]

where:

- |                  |   |
|------------------|---|
| <i>structure</i> | - IDL structure name  |
| <i>dataset</i>   | - indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. If dataset is an array then all datasets will be read into the same structure. |
| <i>number</i>    | - If present, then a total of number datasets will be read and stored in the structure<br><b>Note:</b> If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.                |
| <i>fileid</i>    | - file id of the PFF file containing the data   |
| Structure Labels |   |
| <i>type</i>      | - 4   |
| <i>ndim</i>      | - number of spatial dimensions = 3  |
| <i>block</i>     | - number of blocks = 1  |
| <i>size</i>      | - number of planes according to the following<br>size (0) = number not normal to any axis<br>size (1) = number normal to x<br>size (2) = number normal to y<br>size (3) = number normal to z                                      |

<i>plane(2,3,size(4))</i>	size (4) = Total number of planes
<i>normal(size(4))</i>	- array of the min and max in each dimension
	- number 0, 1, 2, or 3 corresponding to the axis normal to the plane n0, n1, n2, n3 array of indices with planes normal to no axis, x, y, or z axis
<i>slant(3,2,size(0))</i>	- opposite corners of a slant surface
<i>xlab(3)</i>	- spatial labels
<i>tlab</i>	- type label
<i>clab</i>	- dataset comment
<i>file</i>	- file data read from
<i>spare</i>	- integer array (5)

**Examples:**

Read dataset 3 in file id 4 into structure c1

```
READCON, c1, 3, fil=4
```

Read next dataset of the current file into structure anode

```
READCON, anode
```

read datasets 3, 4, 5 into structure c2

```
READCON, c2, 3, 3
```

or

```
READCON, c2, [3,4,5]
```

## READFLD

Reads a field or charge density into an IDL structure.

Procedure options are explained below. PFF dataset type must be 7 N-Dimensional Vector on M-spatial dimensions or 3-Dimensional vector or scalar data.

**format:** READFLD, structure [, dataset] [, FILEID = fileid]

where:

<i>structure</i>	- IDL structure name
<i>dataset</i>	- indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. Note: If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.
<i>fileid</i>	- file id of the PFF file containing the data

**Structure Labels**

<i>type</i>	- Structure type, Field Type = 1
<i>ndim</i>	- number of spatial dimensions
<i>nvect</i>	- number of vector dimensions
<i>block</i>	- number of blocks
<i>two</i>	- two dimensional missing dimension. This is 0 for 3-D data, 1 for yz data, 2 for zx data, 3 for xy data

- slice* - value of the missing dimension, if known.  
Default is zero.
- size* - array of the size of the spatial dimensions  
size = size(ndim, block)
- range* - two dimensional array of the min and max for each block. Block (0) (range (\*,\*, 0) has overall values  
range = range(2, ndim, nblock+1).
- order* - Ordering of blocks in each dimension by initial value of the block, in units of (block -1)  
order = order(ndim, nblock)
- connect* - is zero for single block data  
connect = connect (Npar, nconnect)  
nconnect is the connection number defined below  
Npar is 7 with each value indicating the following:  
0 - plane for the connection 1, 2, 3 (normal direction)  
1 - block with smallest minimum in the normal direction  
2 - block with the minimum equal to the connection plane  
3 - low value of first remaining dimension of the overlap  
4 - high value of first remaining dimension of the overlap  
5 - low value of the last remaining dimension of the overlap  
6 - high value of the last remaining dimension of the overlap  
Example: coordinates x, y, z, blocks 1 and 2 connect at y = 10 and 1 is below 2 then parameter 0 is 2, 1 is 1, 2 is 2, 3 and 4 are the minimum and maximum values of x for the overlap and 5 and 6 are the minimum and maximum values of z for the overlap.
- locx* - array of the location of the first point for each block spatial array: locx = locx(ndim, block+1)  
(first spatial dimension for block m is x and is defined by: x = x1(locx(0, m-1):locx(0, m)-1) )
- locv* - array of the location of the first point for each block vector array: locv = locv(block+1)  
(first vector array for block m is v and is defined by v = reform (v1(locv(m-1):locv(m)-1),size(\*,m-1))
- x1, x2, x3, x4* - spatial arrays. Default is 0
- v1, v2, v3, v4, v5, v6* - vector arrays. Default is 0
- x1lab, x2lab, x3lab, x4lab* -

- v1lab, v2lab, v3lab, v4lab, v5lab, v6lab* - spatial labels. Default is ''
- vector labels, Vector label in the case of NGD data and block label in the case of vector and 3d data.* Default is ''
- tlab* - dataset type label
- clab* - dataset comment label
- file* - file data read from
- spare* - integer array (5)

**Note:** Spatial and vector arrays are reformed to linear arrays and stored in the respective arrays.

Examples:

Read dataset 34 in file id 4 into structure ch5

```
READFLD, ch5, 34, fil=4
```

Read next dataset of the current file into structure efldstart

```
READFLD, efldstart
```

## READGRD

Reads a GRID dataset into an IDL structure.

Procedure options are explained below. PFF dataset type must be NG3 (nonuniform 3-D grid)

**format:** READGRD, structure [, dataset] [, FILEID = fileid]

where:

- structure* - IDL structure name
- dataset* - indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset.  
**Note:** If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.
- fileid* - file id of the PFF file containing the data

Structure Labels

- type* - 3
- ndim* - number of spatial dimensions = 3
- block* - number of blocks
- size* - array of the size of the spatial dimensions  
size = size(ndim, block)
- two* - two dimensional missing dimension. This is 0 for 3d data, 1 for yz data, 2 for zx data, 3 for xy data
- range* - (2, ndim, nblock+1) two dimensional array of the max and min for each block. Block (0) has overall

	values.
<i>locx</i>	- array of the location of the first point for each block spatial array- $locx = locx(ndim, block+1)$ (first spatial dimension for block <i>m</i> is <i>x</i> and is defined by $x = x1(locx(0, m-1): locx(0, m)-1)$ )
<i>x1, x2, x3</i>	- spatial arrays
<i>x1lab, x2lab, x3lab</i>	- spatial labels
<i>blab</i>	- array of block labels
<i>tlab</i>	- type label
<i>clab</i>	-dataset comment
<i>file</i>	- file data read from
<i>spare</i>	- integer array (5)

## Examples:

Read dataset 34 in file id 4 into structure g5

```
READGRD, g5, 34, fil=4
```

Read next dataset of the current file into structure gstart

```
READGRD, gstart
```

**READHDR**

Reads the header of a pff dataset and prints the parameters. Procedure options are explained below.

**format:** READHDR, dataset, ier, fileid = fid, pfftype = pfftype, aptype= aptype, version = version, dlength = dlength, resint = resint, tlabel = tlabel, ntype = ntype, clabel = clabel, ncomment = ncomment, quiet = quiet

where:

<i>dataset</i>	- indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. <b>Note:</b> If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.
<i>ier</i>	- Error indicator

Keywords:

<i>fileid</i>	- File id of the PFF file containing the dataset Default is the current file
<i>pfftype</i>	- PFF dataset type
<i>aptype</i>	- Application dataset type
<i>version</i>	- Dataset version number
<i>dlength</i>	- Dataset length in 16 bit words

- resint* - Number of non-default reserved integers
- tlabel* - Dataset type label
- ntype* - Number of characters in the type label  
(only 360 characters may be read by tlabel)
- clabel* - Dataset comment label
- ncomment* - Number of characters in the comment label  
(only 360 characters may be read by clabel)
- quiet* - If present and not zero, do not print any data. This is ignored if the dataset comment or dataset type labels are larger than the available buffer.

**Examples:**

Read dataset 34 header in file id 4 and store the dataset comment in com

```
READHDR, 34, fil=4, c = com
```

Read header of the next dataset of the current file

```
READHDR
```

**READHIST**

Read a unformatted history file from Quicksilver

```
format: READHIST [, FILENAME] [, WDFARRAY] [, /ALLDATA] [, NARRAY = narray]
        [, /HELPME]
```

where:

- FILENAME* - Name of the file. If missing or equal to 0, then DIALOG\_PICKFILE will be used.
- WDFARRAY* - WDF array number or an array of numbers. Default = 1. If more arrays are selected than contained in WDFARRAY then sequential numbers will be used. MAXWDFARRAY will not be increased so some arrays could remain unsaved if a sufficient number of arrays are not available.
- ALLDATA* - If set, the entire file will be stored. If ALLDATA is a string or string array then GET\_MATCH will be used to select datasets. Default : Allow selection from a widget list.
- NARRAY* - Number of datasets read into wdf arrays
- IGNORE* - READHIST expects an integer number of 512 byte records. If IGNORE is set, READHIST truncates the data to an integer number of records.
- HELPME* - If set, print a help message.

Outputs:

The selected WDF arrays will be stored.

**Restrictions:**

This will read files generated on the Intel and UNIX machines but has not been verified on MAC's and DEC machines, but should also work there.

**Procedures:**

1. Open the file and determine if byte swapping is required, the total length of the file, and the total length of the header.
2. Close and reopen and read the string portion of the header
3. Close and reopen the file and read the data
4. Determine the desired datasets
5. Store the data.

**File format:**

Word 0:	Fortran REcord Length
Word 1:	NHISTW (# of history signals)
Word 2:	TIMSTP (simulation time step)
Word 3-5:	SKPHIS (1-3) (history do-list) (t0 = SKPHIS(1)*TIMSTP, dt = SKPHIS(3)* TIMSTP)
Word 6:	PLNAM (# of characters in title)
Word 7 -(6+NHISTW*PLNAM)	ASCII format to NAMHIS(1 - NHISTW)
Additional	Floating point data to the first -1

**Examples:**

Read the data from qcks.his and store in WDF arrays beginning at 4

```
READHIST, 'qcks.his', 4, /all
```

Read the data from a file determine from a GUI and store in arrays beginning in WDF array 1

```
READHIST
```

Read the data from a file with in string 'current' and '^v' and store the arrays beginning in WDF array 1

```
READHIST, 0, 1, a= ['current', '^v']
```

**READIFL**

Reads an integer float list into three arrays. Procedure options are explained below. PFF dataset type must be IFL

**format:** READIFL, iarray, flist, farray [, dataset] [, FILEID = fileid]  
*where:*

<i>iarray</i>	- integer array
<i>flist</i>	- float list
<i>farray</i>	- floating point array
<i>dataset</i>	- indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. <b>Note:</b> If dataset is a string, a leading `^` indicates that the string begins at the start of the dataset comment.
<i>fileid</i>	- file id of the PFF file containing the data

**Examples:**

Read dataset 34 in file id 4 three arrays

```
READIFL, iarray, flist, farray, 34, fil=4
```

Read next dataset of the current file into a, b, c

```
READIFL, a, b, c
```

**READIMAGE**

Read an ascii image file

```
format: READIMAGE, str [, file] [, NPOINT = npoint] [, NCOL =ncol]
```

where:

<i>str</i>	- Structure array number
<i>file</i>	- file name
<i>NPOINT</i>	- Number of points read or to be read. If none 4096 points, set npoints to the value
<i>NCOL</i>	- Number of columns [Default = 4]

Examples:

Read an image into structure 3. Pick the file from a widget.

```
READIMAGE, 3
```

**READNF3**

Reads a field or charge density into an IDL structure. Procedure options are explained below. Current version limited to 20 blocks by the dimensions in pff\_slrnf3. PFF dataset type must be 3: NF3 = Scalar data on a non uniform grid

```
format: READNF3, STRUCT [, dataset] [, FILEID = fileid] [, STRINGLEN = stringlen]
[, ARRAY = array] [, XARRAY= xarray] [, YARRAY= yarray] [, ZARRAY= zarray]
[APPTYP = apptyp] [, LOCX = locx] [, LOCV = locv] [, CLABEL = clabel]
[, TLABEL = tlabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, ZLABEL = zlabel]
[, BLABEL = blabel] [, NAMEFILE = namefile] [, PFFTYP = pfftyp]
[, SPARE = spare]
```

where:

<i>STRUCT</i>	- IDL structure name
---------------	----------------------

<i>dataset</i>	- indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. <b>Note:</b> If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.
<i>FILEID</i>	- file id of the PFF file containing the data
<i>STRINGLEN</i>	- Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 64 for each block. (maximum = nblocks*stringlen < 800) Default value is 16 characters for each label
<i>ARRAY</i>	- If array is present and not zero, then STRUCT will be the array corresponding to block = array. If array is less than 1 then array = 1. If array is greater than the number of blocks, array = nblock. If array is present and zero, it will contain the vector array.
<i>XARRAY</i>	- Values for the first dimension
<i>YARRAY</i>	- Values for the second dimension
<i>ZARRAY</i>	- Values for the third dimension
<i>APPTYP</i>	- Application dataset type
<i>SPARE</i>	- Spare (5, nblock) spare array for each block
<i>X,Y,Z, LABEL</i>	- Labels for x, y, z, and block
Structure Labels	
<i>type</i>	- 1
<i>ndim</i>	- number of spatial dimensions
<i>nvect</i>	- number of vector dimensions
<i>block</i>	- number of blocks
<i>two</i>	- two dimensional missing dimension this is 0 for 3d data, 1 for yz data, 2 for zx data, 3 for xy data
<i>slice</i>	- value of the missing dimension, if known. Default is zero.
<i>size</i>	- array of the size of the spatial dimensions size = size(ndim, block)
<i>range</i>	- (2, ndim, nblock+1) two dimensional array of the max and min for each block. Block (0) has overall values.
<i>order</i>	- (ndim, nblock) Ordering of blocks in each dimension by initial value of the block, in units of block -1
<i>locx</i>	- array of the location of the first point for each block spatial array- locx = locx(ndim, block+1) (first spatial dimension for block m is x and is defined by x = x1(locx(0, m-1): locx(0, m)-1) )
<i>locv</i>	- array of the location of the first point for each block vector array- locv = locv(block+1) (first vector array

for block  $m$  is  $v$  and is defined by  $v = \text{reform}(v1(\text{locv}(m-1):\text{locv}(m)-1), \text{size}(*, m-1))$

$x1, x2, \dots, x(\text{ndim})$  - spatial arrays  
 $v1, v2, \dots, v(\text{nvect})$  - vector arrays  
 $x1\text{lab}, x2\text{lab}, \dots, x(\text{ndim})\text{lab}$  - spatial labels  
 $v1\text{lab}, v2\text{lab}, \dots, v(\text{nvect})\text{lab}$  - vector labels, Vector label in the case of NGD data and block label in the case of vector and 3d data

$t\text{lab}$  - type label  
 $c\text{lab}$  - dataset comment  
 $\text{file}$  - file data read from  
 $\text{spare}$  - integer array (5)

**Note:** If  $\text{block} > 1$  spatial and vector arrays are reformed to linear arrays and stored in the respective arrays.

#### Examples:

Read dataset 34 in file id 4 into structure  $\text{ch5}$

```
READNF3, ch5, 34, fil = 4
```

Read current dataset of the current file into structure  $\text{efldstart}$

```
READNF3, efldstart
```

Read the current dataset into an array  $\text{image}$ , and put the  $x$  and  $y$  coordinates into  $\text{xar}$  and  $\text{yar}$

```
READNF3, image, /arr, xar = xar, yar = yar
```

## READNGD

Reads a field or charge density into an IDL structure. Procedure options are explained below. Current version limited to 20 blocks by the dimensions in  $\text{pff\_slrngd}$ . PFF dataset type must be 7: NGD = N-Dimensional Vector on M-spatial dimensions

**Note:** TWO QUICK data is stored as 3D data. The spatial coordinates are assumed to be the two coordinates following the normal direction in a right hand sense. ie, If  $\text{ispare}(1) = 2$  then the coordinates are ZX.

**Note:** If  $\text{ispare}(1) = 2$  and two dimensions are passed

or

If  $\text{ispare}(1)$  is 2 and three dimensions are passed with the second being degenerate then the field data will be transposed.

```
format: READNGD, structure [, dataset] [, FILEID = fileid] [, WDFARRAY = wdfarray]
        [, STRINGLEN = stringlen] [, XZDATA = xzdata]
```

where:

$\text{structure}$  - IDL structure name  
 $\text{dataset}$  - indicates a PFF dataset number or is a search string for dataset comments. If  $\text{dataset}$  is missing or zero, then it is set to the current dataset.

**Note:** If  $\text{dataset}$  is a string, a leading '^' indicates that the string begins at the start of the dataset comment.

<i>XZDATA</i>	- If set, then no transpose will be done for <code>ispare(1) = 2</code> . Recall from above that the data is assumed to be <code>zxdata</code> if the second spare value is 2.
<i>FILEID</i>	- file id of the PFF file containing the data
<i>STRINGLEN</i>	- Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 256 for each Default value is 32 characters for each label
<i>WDFARRAY</i>	- If keyword is set (not equal to zero), return a wdf structure.
<b>Structure Labels</b>	
<i>type</i>	- 1
<i>ndim</i>	- number of spatial dimensions
<i>nvect</i>	- number of vector dimensions
<i>block</i>	- number of blocks
<i>two</i>	- two dimensional missing dimension this is 0 for 3d data, 1 for yz data, 2 for zx data, 3 for xy data
<i>slice</i>	- value of the missing dimension, if known. Default is zero.
<i>size</i>	- array of the size of the spatial dimensions <code>size = size(ndim, block)</code>
<i>range</i>	- (2, ndim, nblock+1) two dimensional array of the max and min for each block. Block (0) has overall values.
<i>order</i>	- (ndim, nblock) Ordering of blocks in each dimension by initial value of the block, in units of block -1
<i>locx</i>	- array of the location of the first point for each block spatial array- <code>locx = locx(ndim, block+1)</code> (first spatial dimension for block m is x and is defined by <code>x = x1(locx(0, m-1):locx(0, m)-1)</code> )
<i>locv</i>	- array of the location of the first point for each block vector array- <code>locv = locv(block+1)</code> (first vector array for block m is v and is defined by <code>v = reform(v1(locv(m-1):locv(m)-1),size(*,m-1))</code> )
<i>x1, x2, ...x(ndim)</i>	- spatial arrays
<i>v1, v2, ...v(nvect)</i>	- vector arrays
<i>x1lab, x2lab, ...x(ndim)lab</i>	- spatial labels
<i>v1lab, v2lab, ...v(nvect)lab</i>	- vector labels, Vector label in the case of NGD data and block label in the case of vector and 3d data
<i>tlab</i>	- type label
<i>clab</i>	- dataset comment
<i>file</i>	- file data read from
<i>spare</i>	- integer array (5) <b>Note:</b> If <code>block &gt; 1</code> spatial and vector arrays are reformed to linear arrays and stored in the

respective arrays.

Examples:

Read dataset 34 in file id 4 into structure ch5

```
READNGD, ch5, 34, fil = 4
```

Read current dataset of the current file into structure efldstart

```
READNGD, efldstart
```

## READNI3

Reads a field or charge density into an IDL structure. Procedure options are explained below. Current version limited to 20 blocks by the dimensions in pff\_slrni3. PFF dataset type must be 9: NI3 = Scalar data on a non uniform grid

```
format: READNI3, STRUCT [, dataset] [, FILEID = fileid] [, STRINGLEN = stringlen]
[, ARRAY = array] [, XARRAY= xarray] [, YARRAY= yarray] [, ZARRAY= zarray]
[APPTYP = apptyp] [, LOCX = locx] [, LOCV = locv] [, CLABEL = clabel]
[, TLABEL = tlabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, ZLABEL = zlabel]
[, BLABEL = blabel] [, NAMEFILE = namefile] [, PFFTYP = pfftyp]
[, SPARE = spare]
```

where:

<i>STRUCT</i>	- IDL structure name
<i>dataset</i>	- indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. <b>Note:</b> If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.
<i>FILEID</i>	- file id of the PFF file containing the data
<i>STRINGLEN</i>	- Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 64 for each block. (maximum = nblocks*stringlen < 800) Default value is 16 characters for each label
<i>ARRAY</i>	- If array is present and not zero, then STRUCT will be the array corresponding to block = array. If array is less than 1 then array = 1. If array is greater than the number of blocks, array = nblock. If array is present and zero, it will contain the vector array.
<i>XARRAY</i>	- Values for the first dimension
<i>YARRAY</i>	- Values for the second dimension
<i>ZARRAY</i>	- Values for the third dimension
<i>APPTYP</i>	- Application dataset type
<i>SPARE</i>	- Spare (5, nblock) spare array for each block
<i>X,Y,Z, BLABEL</i>	- Labels for x, y, z, and block
Structure Labels	
<i>type</i>	- 1
<i>ndim</i>	- number of spatial dimensions

<i>nvect</i>	- number of vector dimensions
<i>block</i>	- number of blocks
<i>two</i>	- two dimensional missing dimension this is 0 for 3d data, 1 for yz data, 2 for zx data, 3 for xy data
<i>slice</i>	- value of the missing dimension, if known. Default is zero.
<i>size</i>	- array of the size of the spatial dimensions size = size(ndim, block)
<i>range</i>	- (2, ndim, nblock+1) two dimensional array of the max and min for each block. Block (0) has overall values.
<i>order</i>	- (ndim, nblock) Ordering of blocks in each dimension by initial value of the block, in units of block -1
<i>locx</i>	- array of the location of the first point for each block spatial array- locx = locx(ndim, block+1) (first spatial dimension for block m is x and is defined by $x = x1(\text{locx}(0, m-1) : \text{locx}(0, m)-1)$ )
<i>locv</i>	- array of the location of the first point for each block vector array- locv = locv(block+1) (first vector array for block m is v and is defined by $v = \text{reform}(v1(\text{locv}(m-1) : \text{locv}(m)-1), \text{size}(*, m-1))$ )
<i>x1, x2, ...x(ndim)</i>	- spatial arrays
<i>v1, v2, ...v(nvect)</i>	- vector arrays
<i>x1lab, x2lab, ...x(ndim)lab</i>	- spatial labels
<i>v1lab, v2lab, ...v(nvect)lab</i>	- vector labels, Vector label in the case of NGD data and block label in the case of vector and 3d data
<i>tlab</i>	- type label
<i>clab</i>	- dataset comment
<i>file</i>	- file data read from
<i>spare</i>	- integer array (5)

**Note:** If block > 1 spatial and vector arrays are reformed to linear arrays and stored in the respective arrays.

**Examples:**

Read dataset 34 in file id 4 into structure ch5

```
READNI3, ch5, 34, fil = 4
```

Read current dataset of the current file into structure efldstart

```
READNI3, efldstart
```

Read the current dataset into an array image, and put the x and y coordinates into xar and yar

```
READNI3, image, /arr, xar = xar, yar = yar
```

**READNV3**

Reads a field or charge density into an IDL structure. Procedure options are

explained below. Current version limited to 20 blocks by the dimensions in pff\_slrnv3. PFF dataset type must be 4: NV3 = Vector data on a non uniform grid

format: READNV3, STRUCT [, dataset] [, FILEID = fileid] [, STRINGLEN = stringlen]  
 [, ARRAY = array] [, XARRAY= xarray] [, YARRAY= yarray] [, ZARRAY= zarray]  
 [APPTYP = apptyp] [, LOCX = locx] [, LOCV = locv] [, CLABEL = clabel]  
 [, TLABEL = tlabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, ZLABEL = zlabel]  
 [, BLABEL = blabel] [, NAMEFILE = namefile] [, PFFTYP = pfftyp]  
 [, SPARE = spare]

where:

<i>STRUCT</i>	- IDL structure name
<i>dataset</i>	- indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. <b>Note:</b> If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.
<i>FILEID</i>	- file id of the PFF file containing the data
<i>STRINGLEN</i>	- Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 64 for each block. (maximum = nblocks*stringlen < 800) Default value is 16 characters for each label
<i>ARRAY</i>	- If array is present and not zero, then STRUCT will be the array corresponding to block = array. If array is less than 1 then array = 1. If array is greater than the number of blocks, array = nblock. If array is present and zero, it will contain the vector array.
<i>XARRAY</i>	- Values for the first dimension
<i>YARRAY</i>	- Values for the second dimension
<i>ZARRAY</i>	- Values for the third dimension
<i>APPTYP</i>	- Application dataset type
<i>SPARE</i>	- Spare (5, nblock) spare array for each block
<i>X,Y,Z, BLABEL</i>	- Labels for x, y, z, and block
Structure Labels	
<i>type</i>	- 1
<i>ndim</i>	- number of spatial dimensions
<i>nvect</i>	- number of vector dimensions
<i>block</i>	- number of blocks
<i>two</i>	- two dimensional missing dimension this is 0 for 3d data, 1 for yz data, 2 for zx data, 3 for xy data
<i>slice</i>	- value of the missing dimension, if known. Default is zero.
<i>size</i>	- array of the size of the spatial dimensions size = size(ndim, block)
<i>range</i>	- (2, ndim, nblock+1) two dimensional array of the max and min for each block. Block (0) has overall values.

<i>order</i>	- (ndim, nblock) Ordering of blocks in each dimension by initial value of the block, in units of block -1
<i>locx</i>	- array of the location of the first point for each block spatial array- $locx = locx(ndim, block+1)$ (first spatial dimension for block m is x and is defined by $x = x1(locx(0, m-1):locx(0, m)-1)$ )
<i>locv</i>	- array of the location of the first point for each block vector array- $locv = locv(block+1)$ (first vector array for block m is v and is defined by $v = reform(v1(locv(m-1):locv(m)-1),size(*,m-1))$ )
<i>x1, x2, ...x(ndim)</i>	- spatial arrays
<i>v1, v2, ...v(nvect)</i>	- vector arrays
<i>x1lab, x2lab, ...x(ndim)lab</i>	- spatial labels
<i>v1lab, v2lab, ...v(nvect)lab</i>	- vector labels, Vector label in the case of NGD data and block label in the case of vector and 3d data
<i>tlab</i>	- type label
<i>clab</i>	- dataset comment
<i>file</i>	- file data read from
<i>spare</i>	- integer array (5)

**Note:** If block > 1 spatial and vector arrays are reformed to linear arrays and stored in the respective arrays.

**Examples:**

Read dataset 34 in file id 4 into structure ch5

```
READNV3, ch5, 34, fil = 4
```

Read current dataset of the current file into structure efldstart

```
READNV3, efldstart
```

Read the current dataset into an array image, and put the x and y coordinates into xar and yar

```
READNV3, image, /arr, xar = xar, yar = yar
```

**READPAR**

Reads a particle distribution into an IDL structure.

Procedure options are explained below. PFF dataset type must be VTX

```
format: READPAR, structure [, dataset] [, FILEID = fileid]
        [, ELECTRON= ELECTRON] [, PROTON = PROTON]
```

where:

- |                  |  |
|------------------|--|
| <i>structure</i> | - IDL structure name   |
| <i>dataset</i>   | - indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. |
- Note:** If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset

*comment.*  
*fileid* - file id of the PFF file containing the data

**NOTE:** If the number of attributes is three or four then a fifth attribute will be added called 'KE/rest\_mass' and given by the following:

$$bg2 = (v1^2 + v2^2 + v3^2)/c^2$$

$$v5 = \sqrt{bg2 + 1} - 1$$

where the v's refer to the attributes and  $c = 299\,792\,458$

If electron (/e) is present with ANY value

then  $v5 = v5 * 510999.1e-6$  and the label is 'Electron KE (MeV)'

If proton is present and 1 then (note: /p means proton= 1)

(proton = 0 is changed to proton = 1)

$$v5 = v5 * 938.27231$$

or if p is not equal to 1 then  $v5 = v5 * p * 931.49432$

In both cases the label is 'Ion KE (MeV)'

### Structure Labels

*type* - 2  
*ndim* - number of spatial dimensions  
*natt* - number of attribute dimensions  
*block* - number of blocks (=1)  
*nvert* - number of vertices  
*two* - two dimensional missing dimension this is 0 for 3d data, 1 for yz data, 2 for zx data, 3 for xy data slice - value of the missing dimension for two dimensional data; Default=0  
*x1, x2, ...x(ndim)* - spatial arrays  
*v1, v2, ...v(nvect)* - attribute arrays  
*x1lab, x2lab, ...x(ndim)lab* - spatial labels  
*v1lab, v2lab, ...v(nvect)lab* - vector labels, Vector label in the case of NGD data and block label in the case of vector and 3d data  
*tlab* - type label  
*clab* - dataset comment  
*file* - file data read from  
*spare* - integer array (5)

### Examples:

Read dataset 34 in file id 4 into structure ch5

**READPAR, ch5, 34, fil=4**

Read next dataset of the current file into structure partstart

**READPAR, partstart**

## READPFF

Read a PFF dataset. This command allows most dataset types to be read into IDL. The entire array will be read when reading data as well as when reading the header so that the "Reread" option may be used for all multiple reads of the same dataset array. Following a

READ command the dataset pointer will be moved to the next array so that reads of sequential arrays will require no value for the directory entry.

**format:** READPFF [, *fileid*] [, *image*] [, *space*] [, *dataset*] [, *block*]  
 [, TLABEL = *tlabel*] [, CLABEL = *clabel*] [, BLABEL = *blabel*]  
 [, XLABEL = *xlabel*] [, YLABEL = *ylabel*] [, ZLABEL = *zlabel*]  
 [, WLABEL = *wlabel*] [, ATLABEL = *atlabel*] [, LABEL = *label*]  
 [, AP = *apptype*] [, PFF = *pfftype*] [, SPARE = *ispare(5)*]  
 [, MXBLCK = *mxblck*] [, NPOINTS = *npoints*] [, FILE = *file* ]

where:

- fileid* - file id. If it is omitted or zero, the current file is used. If present, the current file pointer will be set to this file.
- image, space* - arrays containing the dataset information defined in Table 2.
- dataset* - dataset directory entry. If it is omitted or zero, the dataset indicated by the current dataset pointer is read. If *dataset* is "R" or "-1" then the previous dataset, stored in a read buffer, will be used to obtain the requested information. ("R" stands for reread.) *dataset* may also be a string or string variable of two or more characters or a single character other than "R". In the case of an n-character string, the first 64 characters of the dataset comments are searched for an exact match. However if the first character is a "^", then only the first n-1 characters of the dataset comment are searched for an exact match to the remaining characters. If this string is unique to dataset, then that dataset is read. If the string is not unique, then all datasets having this string will be listed and the user asked to pick the desired dataset or enter a zero if no dataset is to be read.
- block* - integer indicating the block number. If it is omitted or zero, the first block is read.
- tlabel* - dataset type label - the first 16 characters are stored in the directory
- clabel* - dataset comment - the first 64 characters are stored in the directory
- blabel* - title or label for this block
- xlabel* - first dimension label for this block
- ylabel* - second dimension label for this block
- zlabel* - third dimension label for this block
- wlabel* - fourth dimension label for this block
- atlabel* - attribute labels for vertex data only
- label* - array of all the above labels in the order given (*label(0)* = *tlabel*, *label(1)* = *clabel*, etc.)

- ap* - application dataset type (User parameter)
- pff* - PFF raw dataset type (see section 1.x)
- spare* - spare integer array for the application (User specified integer iarray(5))
- mxblk* - maximum number of blocks in the dataset
- npoints* - array of number of points for each dimension (up to 4 dimensions: *nx*, *ny*, *nz*, *nt*)
- file* - name of the file

**Note:** Text strings are limited to 240 characters for the read command.

**Table 2: Definition of the *Image* and *Space* Arrays in READPFF**

(*nx*, *ny*, *nz* are the number of points in each spatial dimension)

(*m* is the number of spatial dimensions)

(*natt* is the number of attributes)

(*nvert* are the number of vertex values)

(*nfl* are the number of floating point list values)

Dataset Type	<i>Image</i>	<i>Space</i>
1-D Data	Amplitude ( <i>nx</i> )	Grid Array (0 : <i>nx</i> -1)
3-D Data	Amplitude ( <i>nx</i> , <i>ny</i> , <i>nz</i> )	Grid Array (0 : <i>nx</i> + <i>ny</i> + <i>nz</i> -1)
Vector Data	Amplitude ( <i>nx</i> , <i>ny</i> , <i>nz</i> , 3) ( $V_x$ , $V_y$ , $V_z$ at each location)	Grid Array (0 : <i>nx</i> + <i>ny</i> + <i>nz</i> -1)
Vertex Data	Amplitudes ( <i>nvert</i> , <i>natt</i> )	Grid Array (0 : <i>m</i> -1, 0 : <i>nvert</i> -1)
Integer/Float List	Integer List ( <i>nx</i> )	Float List and array (FL = space (0: <i>ny</i> -1), Float array = space ( <i>ny</i> : <i>ny</i> + <i>nz</i> -1) )
N-Dimension Vector on M-Spatial Dims. (NGD)	Amplitude of vector at each point V( <i>nx</i> , <i>ny</i> , <i>nz</i> , <i>nt</i> , <i>natt</i> ) ( <i>natt</i> = Vector dimensionality) ( <i>nvert</i> = Space dimensionality)	Grid Array (0 : <i>nx</i> + <i>ny</i> + <i>nz</i> + <i>nt</i> -1)
Nonuniform 3-D Grid	Real array = [ <i>nx</i> , <i>ny</i> , <i>nz</i> ]	Grid Array (0 : <i>nx</i> + <i>ny</i> + <i>nz</i> -1)

**Examples:**

Read a data array into *image* from the current dataset of the current file

**READPFF, 0, image**

Read a data array into *picture* from block 2 of dataset 6 in the current file.  
The *space* array is not returned.

**READPFF, 0, picture, 0, 6, 2**

Read from the current file the dataset having “mspin” in the first 64 characters of the dataset comment and put the amplitude and spatial arrays into *pin* and *time*. Put the label for the x-axis into the character variable, *xlab*.

```
READPFF, 0, pin, time, 'mspin', x = xlab
```

Read from the file with file id of 34, the current dataset and put the amplitude and spatial arrays into *pinsig* and *time*. Put all the labels into a character array(13), *pinlab*.

```
READPFF, 34, pinsig, time, lab = pinlab
```

Read from the current file the dataset having “pin1” as the first 4 characters in the dataset comment and put the amplitude and spatial arrays into *pin* and *time*. Put all the labels into a character array(13), *lab*.

```
READPFF, 0, pin, time, '^pin1', L = lab
```

## READPOP

Read an ascii pop file

**format:** READPOP, *fname*, *str*

where:

- |              |  |
|--------------|--|
| <i>str</i>   | - Structure array number to store the image<br>Default is Maxstructure   |
| <i>fname</i> | - the file name or a unit number. If a file name then the file is opened and closed. If a number then the file is read directly. If unit is equal to zero then pickfile will be used to determine a file |

Assumes file of the form:

```
$-----$
$ time = 7.40000 sh.
$ (KIMAX, LIMAX) = (60, 152) ; (R,Z) direction.
$ Response used is sfcamrsp taken from file sfcamrsp inputexp/vlb/gs 7.40/ 152/ 60
$ LI =1, (1<=KI<=KIMAX)
0.00000E+00/ 1.04872E-97/ 1.04872E-97/ 1.04872E-97/ 1.04872E-97/
1.04872E-97/ 1.04872E-97/ 1.04872E-97/ 1.04872E-97/ 1.04872E-97/
```

Stores the data in *str*

Examples:

Read an array from unit 2 and store in structure x.

```
READPOP, 2, x
```

Read the image from file 'IMMG74' and store in structure 3

```
READPOP, 'IMMG74', 3
```

Read from a file determined by PICKFILE and store in structure 4

**READPOP, 0, 4**

## READRGA

Read an ascii file into wdf arrays from a residual gas analyzer

**format:** READRGA, str [, file] [, NPOINT = npoint] [, NCOL =ncol]

where:

- str* - Structure array number
- file* - file name
- NPOINT* - Number of points read or to be read. If none 4096 points, set npoints to the value
- NCOL* - Number of columns [Default = 4]

Examples:

Read an image into structure 3. Pick the file from a widget.

**READIMAGE, 3**

## READSIG

This procedure reads a LLNL SIG uniform format ASCII data file. Present version is optimized to file sizes less than 524288. This can be changed with the keyword NP

**format:** READSIG, Filename, Wdf

**READSIG, Wdf**

**READSIG, Filename, Init, Dt, Y-values**

**READSIG, Init, Dt, Y\_Val**

where:

- Filename* - Character string containing the filename
- Wdf* - WDF array number

Output:

- Wdf* - SIG data in WDF array
- init* - Initial data point
- Dt* - Point spacing
- Y\_Val* - Y-values for the data

Keywords:

- NP* - If specified the initial read will use NP points. If undefined or less than 2 then NP = Default = 524288. If not successful, data will be reread but this will take longer. NP will return the actual number of points read including init and dt

*XLABEL, YLABEL, CLABEL* - Only used with 1 or 2 parameters. Dataset

abscissa, ordinate and comment labels. Default value for XLABEL and YLABEL is ' '. Default value for CLABEL is FILENAME.

#### Examples:

Read data1.sig into idl variables and return the number of points in J

```
j=0
```

```
READSIG, 'data1.sig', t0, dt, y, np =j
```

Read data1.sig into WDF array 5 with labels

```
READSIG, ' data1.sig', 5, xlabel = 'History', Ylabel = '# of Particles', Clabel = 'QS  
Particle Creation'
```

## READSTR

Reads a structure dataset into a QS structure array . Procedure options are explained below.

**format:** READSTR, STR, [, dataset][, NDSET] [, FILEID = fileid] [, GRID = GRID]  
[, PARTICLE = PARTICLE] [. CONDUCTOR = CONDUCTOR]  
[, FIELD = FIELD] [, STRUCTURE = STRUCTURE]

where:

- |                  |   |
|------------------|---|
| <i>STR</i>       | - QS structure array number or<br>If str is an undefined variable - STR will be set to the appropriate structure array number -DELVAR will make a variable undefined              |
| <i>dataset</i>   | - indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset.                                |
| <i>ndset</i>     | - used for conductor data to read more than 1 dataset<br><b>Note:</b> If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment. |
| <i>fileid</i>    | - file id of the PFF file containing the data   |
| <i>Particle</i>  | - Indicates a particle dataset. If the Application dataset type is 20, 21, 23 or 24 particle will be set  |
| <i>Conductor</i> | - Indicates a conductor dataset. If the Application dataset type is 2 then conductor will be set  |
| <i>Grid</i>      | - Indicates a grid dataset. If the Application dataset type is 5 then grid will be set  |
| <i>Field</i>     | - Indicates a field dataset. If the Application dataset type is 11, 12, 13, or 14 then Field will be set.<br>Field is the DEFAULT.  |
| <i>Structure</i> | - Structure stored in QS array, STR   |

## Examples:

Read field data in dataset 34 in file id 4 into QS structure array, 5

```
READSTR, 5, 34, fil=4
```

Read next dataset of the current file into QS structure array 6. Assume dataset is particle data.

```
READSTR, 6, /p (< or > if from QS or TQ) readstr, 6
```

Read next dataset of the current file into QS structure array 7. Assume dataset is particle data and return the structure in PAR.

```
READSTR, 7, /p, structure = par
```

Read conductor datasets 2, 3, 4, 5 into QS structure array 7.

```
READSTR, 7, /con, 2, 4
```

**READTEK2**

Read an ascii data file of the following form: This is a “tek plot” file from mach2 old style with data in columns, one variable to a column

TITLE = ‘restart file, 1m shell’

VARIABLES = “x”, “y”, “Ux”, “Uy”, ....., “Mat#”

ZONE T = “Block 1”, I = 93, J = 161, F = POINT

data - block 1

ZONE T = “Block 2”, I =93, J =161, F = POINT

data - block 2

etc

**format:** READTEK2 [, *fname*] [, WDF] [, LABEL = *label*] [, MAXBLOCK = *maxblock*] [, NPOINTS = *np*] [, NARRAY = *narray*] [MAXNP = *maxnp*]

where:

*fname* - the file name  
if a file name then the file is opened and closed at the end. If *fname* is zero or omitted then PICKFILE will be used to get the file name

**Note:** If only one parameter is given and the parameter is a number, then READTEK2 assumes that it is a STR number.

*STR* - Structure array number or an arrays of numbers.  
Default is 1

*NARRAY* - number of arrays read

The following keyword is ignored in this version ie NDIM == 2

*NDIM* - Number of Dimensions - Default = 2

*Group* - Grouping of arrays into datasets.  
Default = [3, 1, 3, 1, 1, 1, 1, 1]

*MAXNP* - Maximum number of points, default current version = 131072

**MAXBLOCK** - Maximum number of blocks, default current version = 8

#### Examples:

Read a file and store the data beginning in WDF array 1

**READTEK2, 0**

or

**READTEK2, 1**

**Note:** Default WDF array is 1 but procedure must be called with at least 1 parameter

Read a file and store the arrays starting in wdfarray 10

**READTEK2, 10**

or

**READTEK2, 0, 10**

Read file 'DAT1234' and store the data in WDF array 1

**READTEK2, 'DAT1234', 1**

or

**READTEK2, 'DAT1234' ; since WDF = 1 is default**

Read a file and store in arrays [1, 3, 5, 7, 9, 11]

**READTEK2, 1+2\*indgen(6)**

**Note:** If there are more than 6 waveforms, additional waveforms will be stored in 12,13, ....

If there are less than 6 waveforms, no data will be stored in the additional WDF arrays.

## READTEK3

Read an ascii data file of the following form: This is a "tek plot" file from mach2 new style with data in output, one variable at a time. ie all the x, then all the y,....

TITLE = 'restart file, 1m shell'

VARIABLES = "x", "y", "Ux", "Uy", ....., "Mat#"

ZONE T = "Block 1", I = 93, J = 161, F = POINT

data - block 1

ZONE T = "Block 2", I = 93, J = 161, F = POINT

data - block 2

etc

**format:** READTEK3 [, *fname*] [, WDF] [, LABEL = label] [, MAXBLOCK = maxblock] [, NPOINTS = np] [, NARRAY = narray] [MAXNP = maxnp]

where:

*fname*

- the file name

if a file name then the file is opened and closed at the end. If *fname* is zero or omitted then PICKFILE will be used to get the file name

**Note:** If only one parameter is given and the parameter is a number, then READTEK3 assumes that it is a STR number.

*STR* - Structure array number or an arrays of numbers.  
Default is 1

*NARRAY* - number of arrays read

The following keyword is ignored in this version ie NDIM == 2

*NDIM* - Number of Dimensions - Default = 2

*Group* - Grouping of arrays into datasets.  
Default = [1, 1, 1, 1, 1]

*MAXNP* - Maximum number of points, default current version = 131072

*MAXBLOCK* - Maximum number of blocks, default current version = 8

Examples:

Read a file and store the data beginning in WDF array 1

```
READTEK3, 0
```

or

```
READTEK3, 1
```

**Note:** Default WDF array is 1 but procedure must be called with at least 1 parameter

Read a file and store the arrays starting in wdfarray 10

```
READTEK3, 10
```

or

```
READTEK3, 0, 10
```

Read file 'DAT1234' and store the data in WDF array 1

```
READTEK3, 'DAT1234', 1
```

or

```
READTEK3, 'DAT1234' ; since WDF = 1 is default
```

Read a file and store in arrays [1, 3, 5, 7, 9, 11]

```
READTEK3, 1+2*indgen(6)
```

**Note:** If there are more than 6 waveforms, additional waveforms will be stored in 12,13, ....

If there are less than 6 waveforms, no data will be stored in the additional WDF arrays.

## READTK

Read an tkdas data file. Code assumes first line contains 1 non-blank label for each column of data. Format consists of tab delimited columns of the form:

**Note:** Each Title may not have spaces in this version.

```
title
```

```
t0
```

delta\_t  
 Npoints  
 y1 (at t0)  
 y2  
 y3  
 .  
 .  
 .  
 yN (at  $t_0 + (N-1) * \text{delta\_t}$ )

**format:** READTK [, *fname*] [, *WDF*] [, *DATA = data1*] [, *LABEL = label1*]  
 [, *NPOINTS = np*] [, *NARRAY = narray*]

where:

*fname* - the file name  
 if a file name then the file is opened and closed at the end. If *fname* is zero or omitted then PICKFILE will be used to get the file name

**Note:** If only one parameter is given and the parameter is a number, then READTEK2 assumes that it is a STR number.

*WDF* - WDF array number or an array of numbers  
 Default is 1  
 If *WDF* is less than 0 and *n\_elements(data) > 0* then no WDF arrays will be stored.

*DATA* - Input data = data(*npoints*+2, *narray*)  
 (Initial time and delta time are first two points.)

*LABEL* - string array of the names of the arrays

*NPOINTS* - number of data values read for each dataset

*NARRAY* - number of arrays read

Examples:

Read a TKDAS file and store the data beginning in WDF array 1

READTK, 0

or

READTK, 1

**Note:** Default WDF array is 1 but procedure must be called with at least 1 parameter

Read a TKDAS file and store the arrays starting in wdfarray 10

READTK, 10

or

READTK, 0, 10

Read file 'DAT1234' and store the data in WDF array 1

READTK, 'DAT1234', 1

or

READTK, 'DAT1234' ; since WDF = 1 is default

Read a file and store in arrays [1, 3, 5, 7, 9, 11]

**READTK, 1+2\*indgen(6)**

**Note:** If there are more than 6 waveforms, additional waveforms will be stored in 12,13, ....

If there are less than 6 waveforms, no data will be stored in the additional WDF arrays.

Read the data from a TKDAS file into an array DATA and store data beginning in wdf array 1.

**READTK, 0, data =data**

To only return the array without storing data in WDF arrays then:

**READTK, -1, data =data**

## READTK2

Read an tkdas data file. Code assumes 2 datasets per file.

**format:** READTK2 [*fname*] [, WDF] [, DATA =data] [, LABEL = label]  
[, NPOINTS = np] [, NARRAY = narray]

where:

*fname* - the file name  
if a file name then the file is opened and closed at the end. If *fname* is zero or omitted then PICKFILE will be used to get the file name

**Note:** If only one parameter is given and the parameter is a number, then READTEK2 assumes that it is a STR number.

*WDF* - WDF array number or an array of numbers  
Default is 1  
If *WDF* is less than 0 and *n\_elements(data) > 0* then no WDF arrays will be stored.

*DATA* - Input data = data(*npoints*+2, *narray*)  
(Initial time and delta time are first two points.)

*LABEL* - string array of the names of the arrays

*NPOINTS* - number of data values read for each dataset

*NARRAY* - number of arrays read

Examples:

Read a TKDAS file and store the data beginning in WDF array 1

**READTK2, 0**

or

**READTK2, 1**

**Note:** Default WDF array is 1 but procedure must be called with at least 1 parameter

Read aTKDAS file and store the arrays starting in wdfarray 10

**READTK2, 10**

or

**READTK2, 0, 10**

Read file 'DAT1234' and store the data in WDF array 1

**READTK2, 'DAT1234', 1**

or

**READTK2, 'DAT1234' ; since WDF = 1 is default**

Read a file and store in arrays [1, 3, 5, 7, 9, 11]

**READTK2, 1+2\*indgen(6)**

**Note:** If there are more than 6 waveforms, additional waveforms will be stored in 12,13, ....

If there are less than 6 waveforms, no data will be stored in the additional WDF arrays.

Read the data from a TKDAS file into an array DATA and store data beginning in wdf array 1.

**READTK2, 0, data =data**

To only return the array without storing data in WDF arrays then:

**READTK2, -1, data =data**

## READUF1

Reads a time history into an IDL structure. Procedure options are explained below. Current version limited to 20 blocks by the dimensions in pff\_slruf1. PFF dataset type must be 1: UF1 = Scalar data on a 1D uniform grid

**format:** **READUF1, STRUCT** [, dataset] [, FILEID = fileid] [, STRINGLEN = stringlen] [, CLABEL = clabel] [, TLABEL = tlabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, BLABEL = blabel] [, NAMEFILE = namefile] [, APTTYP = apttyp] [, ISPARE = ispare] [, WDFarray = wdfarray]

where:

*STRUCT*

- IDL structure name

*dataset*

- indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset.

**Note:** If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.

*FILEID*

- file id of the PFF file containing the data

*STRINGLEN*

- Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 64 for each block. (maximum = nblocks\*stringlen < 800)

Default value is 16 characters for each label

*TLABEL*

- Dataset type label

<i>CLABEL</i>	- Dataset comment label
<i>XLABEL</i>	- Dataset xlabel for each block
<i>BLABEL, YLABEL</i>	- Dataset block label for each block
<i>APPTYP</i>	- Application dataset type
<i>ISPARE</i>	- Spare array
<i>NAMEFILE</i>	- Name of the PFF file
<i>WDFarray</i>	- If keyword is not zero, STRUCT = [X0, DX, Yarray] where X0 is the initial X value, DX is the X spacing and YARRAY are the ordinate values. Only single block data may be returned as a WDFarray. If WDFARRAY is zero or omitted then the data will be returned as a structure with following labels:

## Structure Labels

<i>type</i>	- 6 type for WDF arrays
<i>ndim</i>	- 1l dimension
<i>block</i>	- 1 Maximum number of blocks
<i>range</i>	- 2 element array with the minimum and maximum abscissa values
<i>yrange</i>	- 2 element array with the minimum and maximum ordinate values
<i>np</i>	- number of points zero indicates no data in this array
<i>x0</i>	- Initial point
<i>dx</i>	- Spacing for uniform data; -1 for x-y data
<i>x</i>	- Abscissa values
<i>y</i>	- Ordinate values
<i>xlab</i>	- Abscissa label
<i>ylab</i>	- Ordinate label
<i>locv</i>	- Array of the location of the first point for each block
<i>tlab</i>	- dataset type label
<i>clab</i>	- dataset comment label
<i>file</i>	- file from which the data originated
<i>ispare</i>	- Spare array for the first block

If multiple blocks exist, then some of these will be arrays

## Examples:

Read dataset 34 in file id 4 into structure ch5

```
READUF1, ch5, 34, fil = 4
```

Read current dataset of the current file into structure efldstart

```
READUF1, efldstart
```

Read the current dataset into a WDFarray image

```
READUF1, image, /wdf
```

**READUF3**

Reads a field or charge density into an IDL structure. Procedure options are explained below. Current version limited to 20 blocks by the dimensions in pff\_slruf3. PFF dataset type must be 1: UF3 = Scalar data on a uniform grid

**format:** READUF3, STRUCT [, dataset] [, FILEID = fileid] [, STRINGLEN = stringlen] [, ARRAY = array] [, XARRAY= xarray] [, YARRAY= yarray] [, ZARRAY= zarray] [APPTYP = apptyp] [, DX = dx] [, DY = dy] [, DZ = dz] , X0 = x0] [, Y0 = y0] [, Z0 = z0] [, CLABEL = clabel] [, TLABEL = tlabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, ZLABEL = zlabel] [, BLABEL = blabel] [, NAMEFILE = namefile] [, PFFTYP = pfftyp]

where:

- |                         |  |
|-------------------------|--|
| <i>STRUCT</i>           | - IDL structure name   |
| <i>dataset</i>          | - indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset.<br><b>Note:</b> If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment. |
| <i>FILEID</i>           | - file id of the PFF file containing the data  |
| <i>STRINGLEN</i>        | - Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 64 for each block. (maximum = nblocks*stringlen < 800)<br>Default value is 16 characters for each label   |
| <i>ARRAY</i>            | - If array is present and not zero, then STRUCT will be the array corresponding to block = array. If array is less than 1 then array = 1. If array is greater than the number of blocks, array = nblock. If array is present and zero, it will contain the vector array.       |
| <i>XARRAY</i>           | - Values for the first dimension   |
| <i>YARRAY</i>           | - Values for the second dimension  |
| <i>ZARRAY</i>           | - Values for the third dimension   |
| <i>APPTYP</i>           | - Application dataset type   |
| <i>SPARE</i>            | - Spare (5, nblock) spare array for each block   |
| <i>X,Y,Z, BLABEL</i>    | - Labels for x, y, z, and block  |
| <b>Structure Labels</b> |  |
| <i>type</i>             | - 1  |
| <i>ndim</i>             | - number of spatial dimensions   |
| <i>nvect</i>            | - number of vector dimensions  |
| <i>block</i>            | - number of blocks   |
| <i>two</i>              | - two dimensional missing dimension this is 0 for 3d data, 1 for yz data, 2 for zx data, 3 for xy data   |
| <i>slice</i>            | - value of the missing dimension, if known. Default is zero.   |
| <i>size</i>             | - array of the size of the spatial dimensions size = size(ndim, block)   |

- range* - (2, ndim, nblock+1) two dimensional array of the max and min for each block. Block (0) has overall values.
  - order* - (ndim, nblock) Ordering of blocks in each dimension by initial value of the block, in units of block -1
  - locx* - array of the location of the first point for each block spatial array-  $locx = locx(ndim, block+1)$  (first spatial dimension for block m is x and is defined by  $x = x1(locx(0, m-1):locx(0, m)-1)$ )
  - locv* - array of the location of the first point for each block vector array-  $locv = locv(block+1)$  (first vector array for block m is v and is defined by  $v = reform(v1(locv(m-1):locv(m)-1),size(*,m-1))$ )
  - x1, x2, ...x(ndim)* - spatial arrays
  - v1, v2, ...v(nvect)* - vector arrays
  - x1lab, x2lab, ...x(ndim)lab* - spatial labels
  - v1lab, v2lab, ...v(nvect)lab* - vector labels, Vector label in the case of NGD data and block label in the case of vector and 3d data
  - tlab* - type label
  - clab* - dataset comment
  - file* - file data read from
  - spare* - integer array (5)
- Note:** If block > 1 spatial and vector arrays are reformed to linear arrays and stored in the respective arrays.

**Examples:**

Read dataset 34 in file id 4 into structure ch5

```
READUF3, ch5, 34, fil = 4
```

Read current dataset of the current file into structure efldstart

```
REAUNF3, efldstart
```

Read the current dataset into an array image, and put the x and y coordinates into xar and yar

```
READUF3, image, /arr, xar = xar, yar = yar
```

**READVTX**

Reads a field or charge density into an IDL structure. Procedure options are explained below. Current version limited to 20 blocks by the dimensions in pff\_slrnf3. PFF dataset type must be 5: PFFVTX

```
format: READVTX, ATTRIBUTES [, dataset] [, FILEID = fileid] [, STRINGLEN = stringlen]
        [, NDIM = ndim] [, NATTR = nattr] [, XARRAY= xarray] [, YARRAY= yarray]
        [, ZARRAY= zarray] [, TARRAY = tarray] [APPTYP = apptyp] [, ALABEL = alabel]
        [, CLABEL = clabel] [, TLABEL = tlabel] [, XLABEL = xlabel] [, YLABEL = ylabel]
        [, ZLABEL = zlabel] [, X4LABEL = x4label] [, NAMEFILE = namefile]
        [, PFFTYP = pfftyp] [, SPARE = spare]
```

where:

<i>ATTRIBUTES</i> <i>dataset</i>	- IDL attribute array size = (NVERT, NATTR) - indicates a PFF dataset number or is a search string for dataset comments. If dataset is missing or zero, then it is set to the current dataset. Note: If dataset is a string, a leading '^' indicates that the string begins at the start of the dataset comment.
<i>FILEID</i> <i>STRINGLEN</i>	- file id of the PFF file containing the data - Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 64 for each block. (maximum = nblocks*stringlen < 800) Default value is 16 characters for each label
<i>NDIM</i>	- Number of spatial values
<i>NATTR</i>	- Number of attributes for each vertex
<i>XARRAY</i>	- Values for the first dimension
<i>YARRAY</i>	- Values for the second dimension
<i>ZARRAY</i>	- Values for the third dimension
<i>TARRAY</i>	- Values for the fourth dimension
<i>APPTYP</i>	- Application dataset type
<i>ALABEL</i>	- Labels the attributes
<i>CLABEL</i>	- Comment label
<i>TLABEL</i>	- Type label
<i>SPARE</i>	- Spare (5) spare array for each block
<i>X,Y,ZLABEL</i>	- Labels for x, y, z
<i>X4LABEL</i>	- Label for Tarray (not TLABEL because of type label)
<i>NAMEFILE</i>	- Filename

**Examples:**

Read dataset 34 in file id 4 into structure ch5, put x and y values into xar and yar

```
READVTX, ch5, 34, fil = 4, xar= xar, yar = yar
```

**READWD3**

Read fit parameters from a WDFIT3 fit file

format: READWD3 [, File\_Name]

where:

*File\_Name* - If undefined or zero, PICKFILE will be used  
Data will be read into the WDFIT3 common block

**Examples:**

Read output from a fit and generate a response curve from 0.2 to 4 in WDF array 30. Use pickfile to get the datafile.

```
READWD3 ; read the fit parameter
```

```

data = .2+findgen(500)/499*3.8
scale_regrid, data ; generate the response
put_wdfarray, [.2, 3.8/499, data], 30, clab = 'Response'

```

## REASC

Read an ascii file into wdf arrays.

**format:** reasc, filename, narray,WDF, formatin=format, np = np

where:

- |                 |   |
|-----------------|---|
| <i>filename</i> | - name of the file containing the data<br>If filename is 0, then pickfile will be used to select a file. This file is opened, data is read, and the file is closed after reading data.<br>If filename is a number, then no file is opened but the procedure reads from unit = filename. |
| <i>narray</i>   | - number of arrays in each record   |
| <i>WDF</i>      | - First WDF array or an array of <i>narray</i> WDF arrays. If the file contains labels, they are stored as the dataset comments.  |
| <i>np</i>       | - number of data records in the file. If the format is 1 or 3, then np is read from the file. If np is not specified only 4096 records will be read.  |
| <i>label</i>    | - string array of labels of the narray arrays   |
| <i>format</i>   | - integer indicating the data format<br>0 only data values stored in the file<br>1 number of data points followed by the data values<br>2 narray labels (stored in <i>label</i> ) followed by the data<br>3 DEFAULT - number of points, followed by narray labels and the data.         |

Example:

Read 5 arrays from file 'PFIDL.dat', into WDF arrays 2,3,4,5,6. Assume data format 3.

```
REASC, 'PFIDL.dat', 5, 2
```

Read 3 arrays from file, 'testdata', into WDF arrays 3, 6, 9. Assume data has no header but only 4 columns of data.

```
REASC, 'testdata', 3, [3,6,9], form = 0
```

Read an array from unit 2 and store in wdf array 12. Assume format = 1.

```
REASC, 2, 1, 12, format = 1
```

## RESTORECT

Restore a color table. The first 8 colors and last color are not changed. The intervening colors are interpolated.

**format:** RESTORECT [, *file*] [, FILENAME = *filename*]

where:

- |                 |  |
|-----------------|--|
| <i>file</i>     | - File containing the saved data<br>Default is 'idl_color_table.dat' |
| <i>filename</i> | - Same as <i>file</i> ; alternate form                               |

## RESTOREPFIDL

Restore WDF arrays and structure arrays.

**format:** RESTOREPFIDL [, *file*] [, FILENAME = *filename*] [, \_EXTRA = *extrastuff*]

where:

- |                   |  |
|-------------------|--|
| <i>file</i>       | - File containing the saved data<br>Default is 'pfidlsave.dat' |
| <i>filename</i>   | - Same as <i>file</i> ; alternate form                         |
| <i>extrastuff</i> | - Any keywords accepted by restore                             |

## REVIDA

Read a VIDA file into a wdf array

Format is :

```
xlabel ylabel
x1      y1
...     ....
xn      yn
-999   -999
```

**format:** REVIDA , *fname*, WDF, XYDATA = *xydata*

where:

- |                |   |
|----------------|---|
| <i>fname</i>   | - the file name or a unit number<br>If a file name then the file is opened and closed at the end. If a number then the file is read directly. If unit is equal to zero then pickfile will be used to determine a file |
| <i>WDF</i>     | - WDF array for storage   |
| <i>npoints</i> | - Maximum number of points to be read, Default = 4096<br>Returned with the actual number read   |
| XYDATA         | - If set to zero, then the data will be put on a uniform grid.  |

- Default is XYDATA = 1, ie save as read.
- SKIP** - If a number gt 0, then that many lines will be skipped before rading the labels.

**Examples:**

Read a VIDATA array from unit 2 and store in wdf array 12.

**REVIDA, 2, 1**

Read VIDATA data from a file selected by pickfile and store it in WDF array 10

**REVIDA, 0, 10**

Read VIDATA arrays from file, 'testdata.vida' into WDF array 3

**REVIDA, 'testdata.vida', 3**

**RISET**

Calculate the rise time of a pulse.

XY data is converted to a uniform time base

(See FALLT for the fall time.)

**format:** rise = RISET (WDF [, STARTVALUE = startvalue] [, ENDVALUE = endvalue]  
[, SMOOTH = smooth] [, VALUE = value] [, MAXIMUM = maxamp]  
[, MAXTIME = maxtime])

where:

- rise* - risetime of the pulse between startvalue and endvalue
- WDF* - WDF array for the calculation
- STARTVALUE* - Beginning values as a fraction of the peak.  
Default = 0.10
- ENDVALUE* - End Value as a fraction of the peak.  
Default = 0.9
- SMOOTH* - Width for smoothing of the data  
Default = 5
- VALUE* - Vector with actual times for start and end values
- MAXIMUM* - Maximum value used to determine start and end value
- MAXTIME* - Time at maximum value

**Examples:**

Print, the 10% to 90% rise time of wdfarray 5.

**print, RISET(5)**

**RMBASE**

Subtract a background signal from one or more WDF arrays. This procedure

requires that the shape of the background signal can be measured accurately and that there be a period of time during which the background signal dominates the composite signal. A least square fit of the measured background with the composite signal is performed to obtain the time correlation of the reference background and the proper amplitude. The reference background is then subtracted from the composite signal.

**format:** `RMBASE, wd1, wd2, wd3 [, wd4] [, BASE = base] [, XR = xr]`  
`[, NOPLOT = noplot] [, TSH = tsh]`

where:

- wd1, wd2* - WDF arrays containing the background signal. *wd1* is a windowed portion of *wd2* during the period when the background signal dominates the composite signals.
- wd3, wd4* - WDF arrays containing the composite signals. If *wd4* is present, all WDF arrays between *wd3* and *wd4* will have the background subtraction.
- base* - specifies a uniform baseline option is to be used for the comparison if nonzero. (see COM, /BASE)
- xr* - specifies a range of abscissa values to be use for the plots
- noplot* - indicates that no plots of the comparisons are desired if nonzero
- tsh* - specifies the waveforms are to be shifted to the timing determined from the comparison with *wd1* if nonzero

Example:

Array 24 contains a measured background signal, array 25 contains only that portion of the signal which dominates the composite signals in arrays 2 through 23. Remove the background signal for the composite signals and time shift each of the composite signals to the timing determined from the comparison with the background. Plot the intermediate steps in the interval of 50 to 150 ns.

`RMBASE, 25, 24, 2, 23, /TSH, xr = [ 50e-9, 150e-9]`

## ROTSTR

Rotate, magnify or demagnify, and/or translate an image. Based on IDL procedure ROT.

**Note:** Image is interpolated so rather than sequential rotations, the angles should be summed and one rotation performed.

**format:** `ROTSTR, STR, ANGLE [, SAVE] [, MAGNIFICATION = magnification]`  
`[, CENTER = center] [, MISSING = missing] [, INTERPOLATION = interpolation]`  
`[, /PIVOT]`

where:

- STR* - The image array to be rotated. This is a field type structure or structure array number. It must have two dimensions and uniform. If it is not uniform, it will be put on a uniform grid.
- ANGLE* - Angle of rotation in degrees CLOCKWISE. (Why?, because of an error in the old ROT)
- SAVE* - Structure array number for the rotated image. If *str* is a structure array number, then the rotated image will be saved as a structure array number. If *str* is a structure, then the rotated image will be saved as a structure.  
Default: If *SAVE* is missing, rotated image will be returned in *STR*.
- MAGNIFICATION* - Magnification/demagnification factor. A value of 1.0 = no change, > 1 is magnification and < 1 is demagnification. Generally this is not used.  
Default: Magnification = 1.0
- CENTER* - ( X, Y) Vector values for the center of rotation.  
Default:  $X = 0.5(x_{max} - x_{min})$   
 $Y = 0.5(y_{max} - y_{min})$   
**Note:** If *PIVOT* = 0, then specifying a center is equal to a translation.
- MISSING* - The data value to substitute for pixels in the output image that map outside the input image.  
Default: *MISSING* = 0.0
- INTERP* - Set this keyword for interpolation type.  
Default: *INTERPOLATION* = 1  
Values:  
0 -Nearest neighbor sampling is used.  
1 -Bilinear interpolation  
2 -"Cubic convolution" interpolation. If this parameter is negative, it specifies the value on the cubic interpolation parameter as described in the *INTERPOLATE* function. Valid ranges are  $-1 \leq \text{Interp} < 0$ . If *INTERPOLATION* = 2 then *INTERP* = -1
- PIVOT* - Setting this keyword causes the image to pivot around the point  $X_0, Y_0$ , so that this point maps into the same point in the output image. If this

keyword is set to 0, then the point X0, Y0 in the input image is mapped into the center of the output image.

Default: PIVOT = 1

ROTSTR, 5, slope1, 7

ROTSTR, 7, slope2 Outputs:

ROTSTR returns a rotated, magnified, and translated version of the input image. Note that the dimensions of the output image are always the same as those of the input image.

Procedure:

The POLY\_2D function is used to translate, scale, and rotate the original image.

Examples:

Rotate the structure 1, 33 degrees and magnify it 1.5 times. Use bilinear interpolation to make the image look nice. Store the result in 1.

**ROTSTR, 1, 33, mag = 1.5 ; (Bilinear is the default)**

Same as above but store the rotated image in 5.

**ROTSTR, 1, 33, 5, mag = 1.5**

Display an image and determine a horizontal line using LINSTR. Rotate the image and make the indicated line horizontal. Assume the original image is in 5 and the result to be in 7

**linstr, 5, slope = slope**

**ROTSTR, 5, slope, 7**

NOTE: If multiple rotations are desired it is better to:

**ROTSTR, 5, slope1 + slope2, 7**

The following requires 2 interpolations (avoid if possible):

**ROTSTR, 5, slope1, 7**

**ROTSTR, 7, slope2**

## RTP

Raise a WDF array to a power.

**Note:** RTP uses multiplication if the power is an integer and uses exponentials and logarithms if the power is not an integer. The result is set to zero if a negative number is raised to a non-integer power. An optional keyword (*exact*) requires the array to be greater than or equal to zero for the calculation.

**Note:** If *wd2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**format:** RTP, wd1, value [, wd2] [, EXACT = exact]

where:

- wd1, wd2* - WDF array numbers  
 If *wd2* is present, then  $wd2 = wd1 \wedge value$   
 else  $wd1 = wd1 \wedge value$
- value* - Power to which array is raised (0.5 is square root)
- exact* - If present and nonzero, operation will be carried out only if *value* is an integer or if *wd1* is not negative.

**Examples:**

Raise array 2 to the 1.5 power and store the result in array 2. Set the result to zero for any negative values of array 2

**RTP, 2, 1.5**

Cube array 4 and store the result in array 6

**RTP, 4, 3, 6**

Take the square root of array 4 and store the result in array 5. Do not perform the calculation if array 4 has any negative values

**RTP, 4, 0.5, 5, /exact**

**RTPSTR**

Raise field structure components to a power. Store the result in the initial field structure array or another structure array. All attribute or vector quantities are raised to the power.

**format: RTPSTR, STR1, VALUE, [, STR3] [, /EXACT] [, CLABEL = clabel]**

where:

- STR1* - array number of the first field structure array
- VALUE* - Power to which the first array is raised
- STR3* - array number for the result, if not specified the result is stored in *STR1* (the first array)
- EXACT* - If present and not zero, aborts the calculation if  $STR1 < 0$  and value is a real. This is ignored if *VALUE* is an integer.
- CLABEL* - Optional label for the resultant array

**RESTRICTIONS:**

*STR1* array must be defined and have valid data

*VALUE* must be defined

**PROCEDURE:**

If *VALUE* is an integer then the result is calculated

Result is  $STR1 \wedge VALUE$

Each attribute, or vector component, of *STR1* is raised to the power

If *VALUE* is not an integer then the result is calculated

Result is  $\exp(\text{VALUE} * \log(\text{STR1}))$  if  $\text{STR1} > 0$   
 or Result is 0.0 if  $\text{STR1} < 0.0$

In either case RESULT is stored in STR3, if specified, or STR1

#### EXAMPLES:

Raise structure 2 to the 1.5 power and store the result in structure 3. Set any negative values in array 2 to zero

**RTPSTR, 2, 1.5, 3**

Raise structure 2 to the 0.5 power and store the result in structure 3. Do not do the calculation if any values are  $< 0.0$

**RTPSTR, 2, 0.5, 3, /exact**

Raise structure 2 to the 3 power and store the result back into 2

**RTPSTR, 2, 3**

## S2I

This function returns a structure from the structure arrays

**format: S2I, str, i**

where:

*str* - the returned structure  
*i* - structure array number

Example:

Obtain structure 3 into variable, f1

**S2I, f1,3**

## SAVECT

Save the R, G, B values of a color table.

**format: SAVECT [, FILE = file]**

where:

*file* - File containing the saved data  
 Default is 'idl\_color\_table.dat'

## SAVEPFIDL

Save the data stored in WDF arrays and in structure arrays. A maximum of 82 arrays of each type will be saved.

**format: SAVEPFIDL [, FILE = file] [\_EXTRA = extrastuff]**

where:

*file* - File containing the saved data  
 Default is 'pfidlsave.dat'

*\_EXTRA* - Any additional keywords recognized by SAVE

Example:

Save the data from the current session

**SAVEPFIDL**

Save the data from the current session in file 'qs5581.sav'

**SAVEPFIDL, file = 'qs5581.sav'**

## SCALE\_REGRID

Fortran routine which will scale data according to a polynomial equation with specified coefficients.

**Note:** See also PFF\_REGRID. Optionally the routine will average points to reduce the size of the array. Optionally the routine will output the data to a PFF file.

**Note:** If fit occurs in the same idl session, then parameters will be taken from the WDFIT3 common block. READWD3 routine will also fill the WDFIT3 common block.

**format:** SCALE\_REGRID, DATA [, X] [, Y] [, NEWX] [, NEWY] [, BREAK = break] [, C1 = c1] [, C2 = c2] [, C3 = c3] [, FUDGE = fudge] [, /EXPONENTIAL] [, /WRITEPFF] [, /REFORM\_DATA] [, /ZERO\_ORIGIN] [, VALID = valid] [, DEFAULT\_VAL = default\_val] [, TLABEL = tlabel] [, CLABEL = clabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, ZLABEL = zlabel] [, BLABEL = blabel] [, IER = ier]

where:

- DATA* - Array to be scaled. May be one or two dimensions. Data must be a float array. Data will be changed. If DATA = DATA(NX, NY) then the returned value will be DATA = DATA (NX/NEWX, NY/NEWY)  
**Note:** NY may equal 1  
If DATA is not 2 dimensional then newy = 1
- X, Y* - Optional X and Y parameters for data. Two element vector with initial X value and X spacing.  
Default: X, Y = [0,1] These will be modified depending on NEWX, NEWY.
- NEWX, NEWY* - Used to rescale X or Y.  
NEWX points in the X direction and NEWY points in the Y direction will be averaged to obtain a new average value.  
Default: NEWX, NEWY = 1L (No averaging of data)
- ZERO\_ORIGIN* - If ZERO\_ORIGIN is set then X(0) = 0.0 and Y(0) = 0.0. Otherwise:  
 $x(0) = x(0) + x(1)*newx/2$   
 $y(0) = y(0) + y(1)*newy/2$
- VALID* - Two element vector with minimum and maximum

- values for the scaling. DATA values outside these limits will be set to the values of default/  
Default: VALID = [min)data, mix = mdata), mdata]
- DEFAULT\_VAL* - If DATA is outside the range set by valid, then these are the values to which DATA will be set.  
If data < valid(0) then data = default\_val(0)  
If data > valid(1) then data = default\_val(1)  
Default values are: DEFAULT\_VAL = scale\_regrid (VALID) unless EXPONENTIAL is set. If exponentail is set, DEFAULT\_VAL(0) = 0.0
- BREAK* - A vector of one or two elements indicating the positions where the C1 coefficients go to the C2 coefficients go to the C3 coefficients The second element, if it exists, is where the C2 coefficients go to the C3 coefficients. For three regions to fit, C1, C2 and C3 must each have two or more elements and BREAK must have two elements.
- FUDGE* - Factor used to insure the function is continuous between polynomial fits. Ignored if N\_Elements(c2) lt 2. This is, if data has no breaks fudge is not used.  
Default: FUDGE = 0.05 of Valid  
Fudge details:  
Let x be the value of image and y the scaled value  
 $y = \text{poly}(x, c1)$   
 $\text{range1} = \text{break}(1) - \text{valid}(1)$   
if  $(\text{break}(1) - x) / \text{range1}$  is less than  $\text{fudge} / 2$  then y is modified  
 $y = y + \text{delta1} * (1 - (\text{break}(1) - x) / (\text{fudge} * \text{range1}))$   
where  $\text{delta1} = \text{poly}(\text{break}(1), c2) - \text{poly}(\text{break}(1), c1)$   
C2 is also modified
- EXPONENTIAL* - If set then after the scaling the data will be exponentiated.  
Default: Exponentail = 0  
If exponentail = 0 then data = poly(data, C)  
If exponential = 1 then data = exp(poly(data, C))
- WRITEPFF* - If set, data will be reformatted to:  
data = data ;ifnewx = 1, newy = 1  
else data = data(nx/newx, ny/newy)  
Default: If WRITEPFF is set, data will not be reformatted. If WRITEPFF is not set, data will be rformatted.

The following are used only if WRITEPFF is set

Default value for all these is: ‘ ‘

<i>TLABEL</i>	- Dataset TYPE label
<i>CLABEL</i>	- Dataset COMMENT label
<i>XLABEL</i>	- Dataset X label
<i>YLABEL</i>	- Dataset Y label
<i>ZLABEL</i>	- Dataset Z label
<i>BLABEL</i>	- Dataset BLOCK label

Example:

Obtain polynomial scaling of the form:

$$y = 45 + 7x + 3x^2$$

for 501 x values in the range 0 to 10 and plot the results

```
image = findgen(501)*(10.0/500)
xx = image           ;need to save a copy for the plot
SCALE_REGRID, image, c1[45, 7, 3]
plot,xx, image
```

## SCANF

Use XINTERANIMATE to build an animation of a DO-list of datasets of FIELD snapshots.

format: **SCANF**, **FLD**, **NORM**, **NORM\_VAL**, **Start** [, **Stop**] [, **Skip**] [, **CON=CON**] [, **SCALE=SCALE**] [, **INDEX=INDEX**] [, **XSIZE = xsize**] [, **YSIZE = ysize**] [, **WINNUM=WINNUM**] [**\_EXTRA = extrastuff**]

where:

<i>FLD</i>	- Field component
<i>NORM</i>	- Plane normal for PLOTFLD
<i>NORM_VAL</i>	- Value of normal ordinate (PLOTFLD argument)
<i>Start \</i>	
<i>Stop</i>	- DO list for frames in file
<i>Skip /</i>	

Default Value for Stop is number of datasets in current PFF file

If Stop le start then stop = ndspff()

If Skip le 0 or undefined then skip =1

<i>CON</i>	- Conductor structure
<i>SCALE</i>	- Scale factor for field (QHRE argument)
<i>INDEX</i>	- Index of plane (PLOTFLD option)
<i>WINNUM</i>	- IDL window number
<i>XSIZE</i>	- X Size of the window; Default = 450
<i>YSIZE</i>	- Y Size of the window; Default = 300

**Example:**

Make a movie using datasets 215 to 228, with levels from 0 to 8e8 and put a thermometer on the plot in the theta plane at theta = 0. Plot the square root of the sum of the squares of the vector components.

```
SCANF, [1, 2, 3], 2, 0, 215, 1, lev = [0, 8e8], /ther
```

**SCANGRD**

Use XINTERANIMATE to build an animation of a DO-list of slices of a GRID or CONDUCTOR snapshot.

```
format: SCANGRD, STR, NORM, Start [, Stop] [, Skip] [, CON=CON]
        [, INDEX=INDEX ] [, XSIZE = xsize] [, YSIZE = ysize]
        [, WINNUM=WINNUM ] [_EXTRA = extrastuff]
```

where:

*STR* - Structure array value or field structure

*NORM* - Plane normal for PLOTFLD

*Start \*

*Stop* - DO list for frames in file

*Skip /*

Default Value for Stop is number of planes in normal direction

If Stop le start then stop = number of indices in NORM direction

If keyword\_set(index) then

If Skip le 0 or undefined then skip =1

*CON* - Conductor structure

*VALUES* - If present then START, STOP and SKIP are ignored. VALUES is an array of normal values.

*INDEX* - Index of plane (PLOTFLD option)

*WINNUM* - IDL window number

*XSIZE* - X Size of the window; Default = 450

*YSIZE* - Y Size of the window; Default = 300

**Example:**

Make a movie using structure array 4 starting at -.1 to 1.1 in increments of .05. Use array 5 to overlay the conductors. Make the z axis as the normal direction and show theta as a polar plot.

```
SCANGRD, 4, 3, -.1, 1.1, 0.01, con = 5, /polar
```

Make a movie of structure 5 with normal direction of 3 at values of Z given by gz. Overlay a conductor from structure 2

**SCANGRD, 5, 3, values = gz, con = 2**

Make a movie of the conductors in structure 2 reflected about theta = 0. Use the z values from block 1.

**image, 2, 3, ref = -2**

**SCANGRD, 3, 3, val = get\_array(2, 1, 1), /polar**

## SCANP

Use XINTERANIMATE to build an animation of a DO-list of datasets of FIELD snapshots.

**format: SCANP, Xaxis, Yaxis, Weigh, Start [, Stop] [, Skip] [, CON=CON]  
[, SCALE=SCALE ] [, INDEX=INDEX ] [, XSIZE = xsize] [, YSIZE = ysize]  
[, WINNUM=WINNUM ] [\_EXTRA = extrastuff]**

where:

- Xaxis* - Particle parameter for the horizontal axis
- Yaxis* - Particle parameter for the vertical axis
- Weight* - Value of normal ordinate (PLOTFLD argument)
- Start \*
- Stop* - DO list for frames in file
- Skip /*

Default Value for Stop is number of datasets in current PFF file

If Stop le start then stop = ndspff()

If Skip le 0 or undefined then skip =1

- CON* - Conductor structure
- SCALE* - Scale factor for field (QHRE argument)
- INDEX* - Index of plane (PLOTFLD option)
- WINNUM* - IDL window number
- XSIZE* - X Size of the window; Default = 450
- YSIZE* - Y Size of the window; Default = 300

Example:

Make a movie using datasets 215 to 228, with axis 3, 1 and color each particle using charge as weight

**SCANP, 3, 1, -4, 215, 228, 1r**

## SCANS

Use XINTERANIMATE to build an animation of a DO-list of slices of a FIELD snapshots.

**format: SCANS, FLD, NORM, Start [, Stop] [, Skip] [, CON=CON]**

[, SCALE=SCALE ] [, INDEX=INDEX ] [, XSIZE = xsize] [, YSIZE = ysize]  
 [, WINNUM=WINNUM ] [\_EXTRA = extrastuff]

where:

<i>FLD</i>	- Field component
<i>NORM</i>	- Plane normal for PLOTFLD
<i>Start \</i>	
<i>Stop</i>	- DO list for frames in file
<i>Skip /</i>	
	Default Value for Stop is number of datasets in current PFF file
	If Stop le start then stop = ndspff()
	If Skip le 0 or undefined then skip =1
<i>CON</i>	- Conductor structure
<i>SCALE</i>	- Scale factor for field (QHRE argument)
<i>INDEX</i>	- Index of plane (PLOTFLD option)
<i>WINNUM</i>	- IDL window number
<i>XSIZE</i>	- X Size of the window; Default = 450
<i>YSIZE</i>	- Y Size of the window; Default = 300

Example:

Make a movie using structure array 4 with every other plane and put a thermometer on the plot and use theta as the slice plane. Plot the square root of the sum of the squares of the vector components. Scale the field between 0 and 8e8

```
SCANS, 4, [1, 2, 3], 2, 1, 0, 2, lev = [0, 8e8], /ther
```

## SCANV

Use XINTERANIMATE to build an animation of a DO-list of datasets of FIELD snapshots.

format: **SCANV, FLD, NORM, NORM\_VAL, Start [, Stop] [, Skip] [, CON=CON]**  
**[, INDEX=INDEX ] [, XSIZE = xsize] [, YSIZE = ysize] [, WORK = work]**  
**[, WINNUM=WINNUM ] [\_EXTRA = extrastuff]**

where:

<i>FLD</i>	- Field components for the vector plot must have two or more elements If 0 or undefined, value = [1, 2]
<i>NORM</i>	- Plane normal for PLOTFLD
<i>NORM_VAL</i>	- Value of normal ordinate (PLOTFLD argument)
<i>Start \</i>	
<i>Stop</i>	- DO list for frames in file

*Skip /*

Default Value for Stop is number of datasets in current PFF file

If Stop le start then stop = ndspff()

If Skip le 0 or undefined then skip =1

*CON*

- Conductor structure

*INDEX*

- Index of plane (PLOTFLD option)

*WINNUM*

- IDL window number

*WORK*

- One or two element vectors used to calculate the components. If only one element then work = [work, work +1]

Default values are [maxstructure -4, maxstructure -3]

*XSIZE*

- X Size of the window; Default = 450

*YSIZE*

- Y Size of the window; Default = 300

*\_EXTRA*

- Valid keyword for PLOTVEC

Example:

Make a movie using datasets 215 to 228, with levels from 0 to 8e8 and put a thermometer on the plot in the theta plane at theta = 0.

```
SCANV, [3, 1], 2, 0, 215, 228, 1, zrange = [0, 8e8], /ther
```

## SET\_DRAWSIZE

Set the default size of draw widgets.

format: **SET\_DRAWSIZE**

**NOTE:**No required parameters. If no keywords, help will be given.

where:

*SHOWDEFAULT* - Show current values

**Note:** Show is executed last; any changes are current

The following are the size in pixels for the draw widgets. If a single value then it is used for XSIZE and YSIZE = ratio\*XSIZE. If two values then they are [XSIZE, YSIZE]

*ratio* - Ratio of YSIZE to XSIZE -- YSIZE = ratio\*XSIZE

*BIGSIZE* - Set default size for TQEDIT, QSEEDIT, WDEDIT

*MEDSIZE* - Set default size for LINSTR, XCURW, WDSPLIT

*DUALSIZE* - Set default size for ZOOMW

*SMALLSIZE* - Set default size for PREVIEW in LINSTR

Example:

Set the size for the preview frame to 450 in X and 400 in Y

```
SET_DRAWSIZE, small = [450, 400]
```

Show current defaults

```
SET_DRAWSIZE, /show
```

## SET\_MAXSTRARRAY

Set the maximum number of structure arrays

**format:** SET\_MAXSTRARRAY, *maxstr*

**where:**

*maxstr* - an integer indicating the maximum number of structure arrays  
 $\text{maxstr} = \text{abs}(\text{long}(\text{maxstr})) > 10$   
 Default maximum is 82

**Note:** If the number of WDFarrays is decreased, data can be lost.

**Example:**

Set the number of structure arrays to 102

SET\_MAXSTRARRAY, 102r

## SET\_MAXWDFARRAY

Set the maximum number of wdf arrays

**format:** SET\_MAXWDFARRAY, *maxwdf*

**where:**

*maxwdf* - an integer indicating the maximum number of WDF arrays.  
 $\text{maxwdfarray} = \text{abs}(\text{long}(\text{maxwdf})) > 10$   
 Default maximum is 82

**Note:** If the number of WDF arrays is decreased, data can be lost.

**Example:**

Set the number of WDF arrays to 102

SET\_MAXWDFARRAY, 102

## SETDEFAULT

Set default conductor/grid parameters

**format:** setdefault [, *ccolor* = *ccolor*] [, *cthick* = *cthick*] [, *gcolor* = *gcolor*]  
 [, *gthick* = *gthick*] [, *transplot* = *transplot*][*directoryhelp* = *dirhelp*]  
 [, *directoryfont* = *dirfont*] [*directorypage* = *dirpage*]

**where:**

*c* and *g* refer to conductor and grid and:

*color* - default color of the conductor must be greater than zero and less than !d.n\_colors 1 to 7 are the basic colors The colors are: red(1), green(2), blue(3), purple(4), orange(5), light blue(6), yellow(7)

*thick* - default line thickness, greater than 0, less than 20

*fcolor* - default color code for contour plots.(PLOTFLD)

*directoryhelp* - 0 no help, 1 print a help for DIRP

*directoryfont* - name of scroll directory font  
*directorypage* - number of lines on a scroll directory

Example:

Set conductor color to red and thickness to 3 times normal and set grid to green and 0.5

**SETDEFAULT, ccolor = 1, cthick = 3, gcolor = 2, gt=0.5**

## SETDIRPFF

This routine sets directory pointer of the current pff file.

format: **SETDIRPFF, dataset**

where

*dataset* - desired dataset pointer value if p1 is undefined or is zero then nothing is changed

Example

set the dataset pointer to dataset 6

**SETDIRPFF, 6**

## SETPFF

Set the current file pointer to file with specified file id

format: **SETPFF, fileid**

where:

*fileid* - file id of desired file

Example:

Set the current file pointer to the file with a file id of 3

**setpff, 3**

## SGIF

Save the current screen information to a file.

format: **SGIF [, FILENAME]**

where:

*FILENAME* - optional filename  
 Default is set by startgif ('idl' is the default value if none is specified ie. idl0001.gif, idl0001.gif ....etc.)

Example:

Send the plots to gif files of the form: movie0001.gif, movie0002.gif, ...

**startgif, file = 'movie'**

**<do a plot>**

**SGIF**

**<do a plot>**

**endgif**

## SHELP

Show the elements of a multidimensional structure or show the structures in the structure common area

**format:** SHELP [, SMIN] [, SMAX] [, /FULL] [, /IDLHELP]

where:

- |                |  |
|----------------|--|
| <i>SMIN</i>    | - First structure array number to be listed or an array of structure array numbers or a QS structure   |
| <i>SMAX</i>    | - Last structure array number  |
| <i>FULL</i>    | - If present and not zero, the full structure array information is printed. If full is zero, only the dataset comment is printed<br>Default: SMIN and SMAX are missing, then FULL = 0 SMAX is missing, then FULL = 1 Otherwise, FULL = 0 |
| <i>IDLHELP</i> | - If present and not zero, the structure component information provided by "help, /structure" is provided. Default is IDLHELP =0   |

Example:

See the contents of all structure arrays

**SHELP**

See the full contents of structure, f12

**SHELP, f12**

See the dataset comment only of structure array, f12

**SHELP, f12, full= 0**

See the full contents of structure arrays 1 through 3

**SHELP, 1, 3, /full**

or

**SHELP, [1,2,3]**

See the contents of structure arrays 10 through 20

**SHELP, 10, 20**

## SHOWPFF

Show all opened PFF files, their access (RE for read, or RW for read/write), their file id and indicate the location of the current file pointer. The current file id, and arrays of the user file ids and filenames can be received as keywords.

**format:** SHOWPFF [, current = current] [, ufid = ufid] [, filename = filename] [, NUMBER= number]

where

- |                 |   |
|-----------------|---|
| <i>current</i>  | - user file id of the current file                |
| <i>ufid</i>     | - array of user file id's                         |
| <i>filename</i> | - Array of file names corresponding to the ufid's |

- number* - number of files open
- quiet* - If present and not zero, send no output to terminal

Example:

Show the status, including file id, of all opened PFF files

**SHOWPFF**

Get the filenames and user file id's for all open files. Do not send any output to the terminal

**SHOWPFF, c= cur, u=u, num= n, fil= fil, /q**

## SLICE

Reduce a three or four dimensional field structure to two dimensions.

**Note:** Four dimensional data is checked for valid data as closely as 3D data.

**format:** **Structure2 = SLICE (Structure1, Field, Kvalue1, Value1, Kvalue2, Value2)**

where:

- Structure1* - Input this is a multidimensional field type structure or a structure array number
- Structure2* - Output this is a two dimensional field structure or a multidimensional field structure with degenerate dimensions.
- Field* - Field component to be sliced

**Note:** Field may be positive or negative;  $\text{Field} = -(\text{abs}(\text{Field}))$

**Note:** If Field is an array, the square root of the sum of the squares will be used.

**Note:** If A has only one attribute (such as an image or a charge density); the this parameter can be omitted.

- KVALUE1,2* - If Field is a two dimensional field then these values are ignored.

If Field is a three or four dimensional field then these values indicate the coordinates which are fixed for the two dimensional plots.

- VALUE1, 2* - If Field is a two dimensional field then these values are ignored.

If Field is a three or four dimensional field then these values indicate the coordinate values for the fixed dimensions in the two dimensional plots. Linear interpolation is used according to the value of VALUE1. If VALUE1 has two elements the average value over the interval is used. If these values span a block boundary, only the block containing the midpoint will be used.

**KEYWORDS:**

*INTEGRATE*

- If present and not zero, integrate the slice.

- The integral will be performed between value1(0) and value1(1)  
 If integrate = 1, a simple integral  
 If integrate = 2, a  $x dx$  integral  
 If integrate = 3, a  $x^2 dx$  integral  
 Limitations:  
 Data must have a single block  
 Two values of value 1 must exist
- OVERPLOT* - If overplot is not zero and kvalue1,2 is undefined, the values will be taken from the last plot.
- BLOCK* - If the field is multi-block, BLOCK can be used to specify one or more blocks to be sliced. BLOCK is an array of one or more values.
- INDEX* - If present and not zero, then the units of VALUEi will be taken as indices of the field array. Valid values are 1 to the maximum number of values in KVALUEi. Care should be exercised with multiple block data.

**Warning:** If INDEX is specified, VALUE1 will be changed to spatial coordinates using the midpoints of the bins.

Outputs:

Structure is modified to produce a two dimensional structure or a structure with n-dimensions and all but two degenerate

EXAMPLE:

Reduce structure f to a two dimensional structure by slicing along the second dimension between 0.0 and 0.4 Save the magnitude of the field vector in structure g.

**g = SLICE (f, [1,2,3], 2, [0.0, 0.4])**

Assume d is a charge density structure with one attribute. Take a slice at  $z = 0.25$  and store in structure array 12.

**i2s, SLICE (d, 3, .25), 12**

or

**i2s, SLICE(d, 1, 3, .25), 12**

## SMO

Boxcar filter of a WDF array.

**format: SMO, wd1 [, wd2] [, WIDTH = width]**

where:

*wd1, wd2*

- WDF array numbers
- If *wd2* is present, then  $wd2 = SMO (wd1)$
- else  $wd1 = SMO (wd1)$

*width*

- width of the boxcar filter; the default value is 5

Examples:

Calculate a box car filter (width = 5) of array 2 and store the result in array 3.

**SMO, 2, 3**

## SP2XYZ

Decompose a spatial array from the READPFF command into its component arrays for 1D, 3D and vector dataset types.

**format:** SP2XYZ, image, space, x [, y] [, z ]

where:

- image* - *image* array from READPFF
- space* - *space* array from READPFF
- x* - one dimensional array for the first spatial dimension of the data
- y* - one dimensional array for the second spatial dimension of the data
- z* - one dimensional array for the third spatial dimension of the data

Examples:

Convert the spatial array, *geom*, of a series of image frames from the array, *picture*, into *x*, *y*, and *time* arrays.

**SP2XYZ, picture, geom, x, y, time**

## SPEC

This procedure the energy values and time of flights for a magnetic spectrometer. The geometry assumes a planar anode on the left. Next a planar gas cell with a three element scattering foil in the center. Gas pressure and type may be different on either side of the scatterer. The spectrometer is assumed to be at an angle to the direct beam and to have one foil at its entrance. The magnetic field in the magnet is assumed constant over the diameter.

**format:** SPEC, fileout, filein, QUIET = quiet, LENGTH = length

where:

- fileout* - Name of file with the output
  - filein* - Input file with spectrometer values
  - QUIET - Range information will be printed if quiet is 0
- Note:** If the ion charge is greater than *z* or less than 0 then all possible charge states will be calculated.  
Default QUIET = 1

Example:

**SPEC**

## SPICT

Save the current screen information to a file.

**format:** SPICT [, FILENAME]

where:

*FILENAME* - optional filename  
 Default is set by startgif ('idl' is the default value if none is specified ie. idl0001.pict, idl0001.pict ....etc.)

Example:

Send the plots to gif files of the form: movie0001.gif, movie0002.gif, ...

startgif, file = 'movie'

<do a plot>

SPICT

<do a plot>

endgif

## STARTEPS

Generate an encapsulated postscript file of the next graphics commands for incorporation into another document; no graphics is sent to the screen. Helvetica font is used.

**Note:** *ENDPOST* must be used to close the file and return to terminal output

**format:** STARTEPS [, FONT = font] [, COLOR= color] [, BOLD = bold]  
 [, PREVIEW=preview] [, FILE=file] [, BITS = bits] [, THICK = thick]  
 [ XYTHICK = xythick]

where:

*font* - font size in points. Default is 18 point  
*color* - no color (black and white plot) is used if color is set to zero. Default is *color* = 1  
*bold* - specifies bold print for the output font if present and nonzero. Default is *bold* = 0  
*preview* - generate a preview frame for the file if present and nonzero. Default is *preview* = 0  
*file* - output file name for the encapsulated postscript output. Default is 'idl.eps'  
*bits* - bits per pixel. Default is 8  
*thick* - plot line thickness. Default is 5  
*xythick* - Axis line thickness. Default is 5

Examples:

Send the next plot to a file called 'QSsummary.eps' with a preview frame and plot a graph.

```

STARTEPS, /preview, file = 'QSsummary.eps
plo, 2, 3, /ov, title = 'Summary of Calculations'
endpost

```

## STARTGIF

Set the screen to generate a plot suitable for generating a GIF file or a PICT file or a TIFF file for inclusion in a word document.

**Note:** ENDGIF must be used to restore the old screen parameters.

True Type fonts are used.

```

format:  STARTGIF [, FILE] [, FONT = font] [, SIZE_CHAR = size_char] [, /NOSWAP]
        [, THICK = thick] [, XYTHICK = xythick] [, FILENAME = file]
        [, _EXTRA = extrastuff]

```

where:

- FILE* - output file name for the GIF or PICT output.  
Default is 'idl0001.gif' or 'idl0001.pict'
- FONT* - Font Name (any True Type font)  
Default = 'Helvetica Bold'  
Default font will be changed to font if font is specified  
**Note:** Use the following to set a different true type font.  
device, set\_font = font, /tt\_font
- NOSWAP* - If set color map will not be switched.  
Default is to issue the command:  
INVERTCT, /RESET
- SIZE\_CHAR* - Width and height for character.  
If no values given - Default = [1, 1.4]\*0.022\*!d.y\_size  
If 1 value given - Default = [1, 1.4\*size\_char  
**Note:** The current font size is stored in !d.x\_ch\_size and !d.y\_ch\_size. These are read only.  
**Note:** Use the following to set another character size.  
device, set\_character\_size = [x\_ch\_size, y\_ch\_size]
- THICK* - specifies the plot line thickness. Default is 2
- XYTHICK* - Specifies the axes thicknesses. Default is 2
- \_EXTRA* - any valid keywords and values for the device command

Example:

Send the plots to gif files of the form: movie0001.gif, movie0002.gif, ...

**Note:** For a PICT file replace SGIF with SPICT. For TIFF replace SGIF with STIFF.

```

    startgif, file = 'movie'
<do a plot>
    SGIF
<do a plot>
    endgif

```

## STARTP

Generate a plot for the default printer.

**Note:** *ENDP* must be used to close the device and return to terminal output.

```

format:  STARTP [, FONT = font] [, COLOR= color] [, BOLD = bold] [, COPY = copy]
          [, FILE=file] [, BITS = bits] [,INTERPOLATE = interpolate] [, THICK = thick]
          [ XYTHICK = xythick] [, NOLOAD = noload] [, PORTRAIT = portrait]
          [, _EXTRA = extrastuff]

```

where:

- FONT* - font thickness. Default = 3  
!p.charthick = font
- COLOR* - black and white output is used if color is set to zero.  
Default is *color* = 0
- BOLD* - specifies bold print for the output if present and nonzero. Default is !p.charthick = 2\*font
- BITS* - bits per pixel. Default is 8
- PORTRAIT* - put output into portrait mode will a full page margin is 0.5 inches; size is 7.5X10 inches.  
Default is landscape
- THICK* - plot line thickness. Default is 5
- XYTHICK* - Axis line thickness. Default is 5
- \_EXTRA* - Any allowed keywords to the device command.
- COPY* - See Set\_Plot - Copy color table from current device to postscript  
Default = 0
- INTERPOLATE* - See Set\_plot - Interpolate current color table to the new device. Default = 0  
**Note:** If X-Y color table is used, these pure colors will be changed
- NOLOAD* - If set, no color table will be loaded.  
Default: LOADXYCT

Note: When using Postscript, Greek characters are accessed using “!9” and the norman font is “!3” for example distance is microns for an axis label would be written: xtitle = ‘Distance (!9m!3m)’

Examples:

Send the next plots to a file called 'QSsummary.ps'

```

STARTP, file = 'QSsummary.ps'
{ plot the stuff}
endp

```

## STARTPCL

Generate a pcl file of all graphics output. No graphics is sent to the screen.

**Note:** *ENDPCL* must be used to close the device and return to terminal output.

```

format: STARTPCL [, FILE] [, FONT = font] [, COLOR= color] [, BOLD = bold]
[, COPY = copy][, FILE=file] [, BITS = bits] [,INTERPOLATE = interpolate] [,
THICK = thick] [ XYTHICK = xythick] [, NOLOAD = noload]
[, PORTRAIT = portrait] [, _EXTRA = extrastuff]

```

where:

- |                    |   |
|--------------------|---|
| <i>file</i>        | - output file name for the postscript output.<br>If the file parameter is used the keyword file is ignored. Default is 'idl.ps'                               |
| <i>charsize</i>    | - Multiplier for character size<br>If set, !p.charsize = charsize   |
| <i>FONT</i>        | - font thickness. Default = 3<br>!p.charthick = font  |
| <i>COLOR</i>       | - black and white output is used if color is set to zero.<br>Default is <i>color</i> = 0  |
| <i>BOLD</i>        | - specifies bold print for the output if present and nonzero. Default is !p.charthick = 2*font  |
| <i>BITS</i>        | - bits per pixel. Default is 8  |
| <i>PORTRAIT</i>    | - put output into portrait mode will a full page margin is 0.5 inches; size is 7.5X10 inches.<br>Default is landscape   |
| <i>THICK</i>       | - plot line thickness. Default is 5   |
| <i>XYTHICK</i>     | - Axis line thickness. Default is 5   |
| <i>_EXTRA</i>      | - Any allowed keywords to the device command.   |
| <i>COPY</i>        | - See Set_Plot - Copy color table from current device to postscript<br>Default = 0  |
| <i>INTERPOLATE</i> | - See Set_plot - Interpolate current color table to the new device. Default = 0<br><b>Note:</b> If X-Y color table is used, these pure colors will be changed |
| <i>NOLOAD</i>      | - If set, no color table will be loaded.<br>Default: LOADXYCT   |

**Note:** When using Postscript, Greek characters are accessed using “!9” and the norman font is “!3” for example distance is microns for an axis label would be written: `xtitle = 'Distance (!9m!3m)'`

Examples:

Send the next plots to a file called 'QSsummary.ps'

```
STARTPCL, file = 'QSsummary.ps
{ plot the stuff}
endpcl
```

## STARTPOST

Generate a postscript file of all the plots; no plots are sent to the screen. Helvetica font is used in landscape orientation.

**Note:** *X2PS* and *PS2X* may be used to switch between Postscript and “X” output to the terminal.

**Note:** *ENDPOST* must be used to close the file before printing.

format: `STARTPOST` [, `FONT = font`] [, `COLOR= color`] [, `BOLD = bold`]  
 [, `FILE=file`] [, `BITS = bits`] [, `PORTRAIT = portrait`] [, `THICK = thick`]  
 [, `XYTHICK = xythick`] [, `_EXTRA = extrastuff`]

where:

- font* - font size in points. Default is 18 point
- color* - no color (black and white plot) is used if color is set to zero. Default is *color = 0*
- bold* - specifies bold print for the output font if present and nonzero. Default is *bold = 0*
- file* - output file name for the postscript output. Default is 'idl.ps'
- bits* - bits per pixel. Default is 8
- portrait* - output is in portrait mode. Margin is 0.5 inches; size is 7.5 wide by 10 inches high.
- thick* - plot line thickness. Default is 5
- xythick* - Axis line thickness. Default is 5
- extrastuff* - Any allowed keywords to the device command.

Examples:

Send the next plots to a file called 'run25.ps'

```
TARTPOST, file = 'run25.ps
{ plot the stuff}
endpost
```

## STIFF

Save the current screen information to a file.

**format:** STIFF [, FILENAME] [, COMPRESSION = compression]

where:

- FILENAME* - optional filename  
 Default is set by startgif ('idl' is the default value if none is specified ie. idl0001.pict, idl0001.pict ....etc.)
- COMPRESSION* - Compression type (see Write\_Tiff)  
 Default = 1  
 If set, all future calls will default to the value set.

Example:

Send the plots to gif files of the form: movie0001.gif, movie0002.gif, ...

startgif, file = 'movie'

<do a plot>

STIFF

<do a plot>

endgif

## STOPPING

Calculate the ion range using the formulation of J. Maenchen

**format:** STOPPING , zion, ein, eout, thick, mat [, press] [, edel] [, ier] [, AION =aion]  
 [, QUIET = quiet] [, ION\_NAME = ion\_name] [, FOIL\_MAT = foil\_mat]

where:

- zion* - z of the ion
- ein* - input energy (Mev) (may be an array)
- eout* - output energy (Mev) (may be an array)
- thick* - foil thickness (microns) (may be an array)
- mat* - z of the foil or one of several compounds listed below:
- |             |                                     |
|-------------|-------------------------------------|
| mat = 0     | vacuum                              |
| mat = 1, 92 | elements with z = mat               |
| mat = 101   | polyethylene: C-H2, rho = 0.92      |
| mat = 102   | mylar: C10-H8-O4, rho = 1.395       |
| mat = 103   | lithium fluoride: Li-F, rho = 2.601 |
| mat = 104   | teflon: C-F2, rho = 2.20            |
| mat = 105   | kimfol: H14-C16-O3, rho = 1.21      |
| mat = 106   | kapton: C22-H10-N2-O4, rho = 1.42   |
| mat = 107   | parylene-n: C8-H8, rho = 1.11       |
| mat = 108   | parylene-c: C8-H7-Cl, rho = 1.28    |
| mat = 109   | parylene-d: C8-H6-Cl2, rho = 1.418  |

	mat = 110	erbium deuteride: Er-D2, rho = 8.64(used H in derivation)
	mat = 111	brass: Cu62-Zn35-Pb3, rho = 8.49
	mat= 112	monel: Fe-Ni67-Cu32, rho = 8.83
	mat = 113	inconel: Cr19-Fe18-Ni53-Mo3-Nb5, rho = 8.19
	mat = 114	LB4: Li0.4-B1.8-O2.9, rho = 2.2
	mat = 115	LB5: Li0.2-B1.8-O2.8, rho = 2.2
<i>press</i>	-	pressure if mat is a gas; default = 1 torr
<i>edel</i>	-	energy loss (ein -eout)
<i>ier</i>	-	error parameter, number of iterations or problemif neg.
<i>AION</i>	-	Ion mass in AMU. If not set, average atomic mass will be used for the stopping.
<i>QUIET</i>	-	if set, no output Default: list the results
<i>ION_MASS</i>		Name of the ion
<i>FOIL_MAT</i>	-	Name of the foil material

**Discussion:**

One of the parameters (ein, eout, thick) must be negative. The negative parameter will be determined. If any of the parameters (ein, eout, thick) are arrays then the other parameters will be made into arrays of equal length using the first value to pad arrays if needed. Only 1 value of zion and nmat are allowed.

**Example:**

Determine the range of Li in Au at 1, 5 and 10 MeV

```
STOPPING, 3, [1, 5, 10], 0, -1, 79
```

or

```
thick = -1 & STOPPING, 3, [1, 5, 10], 0, thick, 79
```

**Note:** The value of THICK will be changed to an arraythicknesses

**STRCOMMENT**

This function returns the struture comment for an array

**format:** com = STRCOMMENT(STR)

*where:*

*STR* - Valid structure number

**Example:**

Print the comment label for structure array 5

```
Print, STRCOMMENT(5)
```

Add the shot number to a comment of structure array 3, where shot number is SHOT

chastr, 3, c = STRCOMMENT(3) + ', SHot' + strtrim(shot, 2)

## STRVALID

**Note:** VALID is a flag to determine the mode of this routine

If keyword (valid) is not set. This function returns an array of valid structure array numbers. Minimum, maximum and a search can be specified

or

If keyword (valid) is set. This function returns an array of valid structure array number from a proposed array of numbers.

**format:** array = STRVALID([STRING] [, Maximum = maximum] [, Minimum = minimum] [, Npoint = npoint] [, Case\_sensitive = case\_sensitive]) [, VALID = valid] [, QUIET = quiet]

where:

If VALID is zero or omitted:

*STRING* - is a search string. Leading and trailing blanks are ignored.

String may be a number. This routine uses:

str = strtrim(string(0), 2)

where str is the actual search string.

For example s = strvalid(24.50) will return a structure with a comment 'Efield, t = 24.50 ns'

If missing, all valid arrays are returned. See GET\_MATCH for available wild card options.

*Minimum* - is the minimum value desired, Default = 1

*Maximum* - is the maximum value desired, Default = maxstructure. Current version will sort maximum and minimum.

*Npoint* - is the number of values returned.

*Case\_sensitive* - If set makes the search case sensitive.

Caution: Case is a special word so use /c or /ca or /cas or /case or more letters but /case will give an error

If VALID is not zero:

*STRING* - an array of possible structure arrays. Must be a byte, integer, long or a float variable. Floats are converted to LONG using ROUND. This array of 1 or more numbers is not optional if VALID is set (ie not equal to zero or omitted). See GET\_MATCH for wild card options

or

a search string. If an array, only the first element is used.

*NPOINT* - is the number of values returned.

*QUIET* - If set, not output will be generated. If not set, messages will be printed if STRING contains

invalid structure arrays. Messages of the form if  
 VALID is a string:  
 “% “ + STRTRIM(VALID, 2) + “ : “ + message  
 or “% INVALID STR: “ + message  
 where message =  
 Structure array number is UNDEFINED  
 or Structure array number must be greater than 0 and  
 </ MX  
 or No data in N structure array where MX is the  
 maximum number of arrays and N is the number of  
 arrays.

For both states of VALID, if NPOINT = 0 then ARRAY = -1

Example:

Find all structure arrays containing data and store in VALS

**VALS = STRVALID**

Find structure arrays with ‘e in inlet’ and print the dataset comments only.

**SHELP, STRVALID(‘ e in inlet’), full = 0**

Determine if strain is a valid WDF array with data. Do not print errors.

**strgood = STRVALID(strin, /valid, /quiet, npoint = npoint)**

Same as above but print an error message beginning with ‘MY\_ADD\_IT’

**strgood = STRVALID(strin, npoint = npoint, valid = ‘MY\_ADD\_IT’)**

## SUB

Subtract a WDF array from another WDF array at corresponding abscissa values. Linear interpolation is used to put the arrays on the same abscissa values. The result only contains values in the region of overlap of the two arrays.

**Note:** If *wd3* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**Note:** *TSH* and *NEWEND* can be used to add zero values to the beginning and end of a WDF array.

**format:** SUB, *wd1*, *wd2* [, *wd3*] [, CLABEL = *clabel*] [, XLABEL = *xlabel*]  
 [, YLABEL = *ylabel*]

where:

- wd1*, *wd2*, *wd3* - WDF array numbers  
 If *wd3* is present, then  $wd3 = wd1 - wd2$   
 else  $wd1 = wd1 - wd2$
- clabel* - Dataset comment for the sum  
 Default is the dataset comment for *wd1*
- xlabel* - X-axis label for the sum  
 Default is the x-axis label for *wd1*
- ylabel* - Y-axis label for the sum  
 Default is the y-axis label for *wd1*

Examples:

Subtract array 2 from array 1 and store the result in array 3

**SUB, 1, 2, 3**

Subtract array 4 from array 3 and store the result in array 3

**SUB, 3, 4**

Subtract a constant from each element of a WDF array.

**Note:** To subtract two constants, use IDL directly. (See last example.)

**format:** SUB, wd1, 0, value [, wd3]

where:

- value* - constant or first element of an array
- wd1, wd3* - WDF array numbers
- If *wd3* is present, then  $wd3 = wd1 - value(0)$
- else  $wd1 = wd1 - value(0)$

Examples:

Subtract 10 from all the elements in array 1 and store the result in array 3

**SUB, 1, 0, 10, 3**

Subtract the variable, *shift*, from all the elements in array 3 and store the result in array 3

**SUB, 3, 0, shift**

Subtract array 4 from arrays 1, 2, 3 and store the results in arrays 5, 6, 7

**for i=1, 3 do SUB, i, 4, 4+i**

Subtract variable, *girls*, from variable, *total*, and store in *boys*

**boys = total - girls**

## SUBSTR

Subtract a field structure array or a constant from another a field structure array Store the result in the initial field structure array or another structure array. All attribute or vector quantities are added.

**format:** SUBSTR, STR1, STR2, [, STR3] [, CLABEL = clabel]

**format:** SUBSTR, STR1, 0, VAR, [, STR3] [, CLABEL = clabel]

where:

- STR1* - array number of the first field structure array
- STR2* - array number of field structure to be subtracted from the first structure
- or
- 0, *VAR* - Variable or constant to be subtracted from the first structure

OPTIONAL INPUT:

*STR3* - array number for the sum, if not specified the sum is stored in STR1 (the first array)

## KEYWORDS:

*CLABEL* - Optional label for the combined structures.

## OUTPUTS:

NONE The difference is stored in a structure array in the structure common block QSCOM.DAT QSSTRCOM.DAT

## RESTRICTIONS:

STR1 array must be defined and have valid data

STR2 or 0 and a Variable must be defined

STR1 and STR2 must both be field arrays and have the same number of vector components.

The SIZE of STR1 and STR2 must be the same

The values of the spatial coordinates must be equal to about  $3e-5$  (ie.  $-\max(\text{abs}(\text{difference}))/\max(\text{value}) \leq 3.1e-5$ )

The check on spatial coordinates is done globally rather than on a block by block basis.

If a structure has three dimensions, but is degenerate in one of them that axis need not be equal.

## PROCEDURE:

If STR2 is specified:

Each attribute, or vector component, of STR1 is subtracted from each attribute of STR2

If 0, VAR is specified:

VAR is subtracted from each attribute of STR1

In either case, the difference is stored in STR3, if specified, or STR1, if STR3 is not specified.

## EXAMPLES:

Subtract structure arrays 2 and 3 and store the result in structure 8

**SUBSTR, 2, 3, 8**

Subtract the variable OFFSET from structure array 2 and store the difference in structure 8

**SUBSTR, 2, 0, offset, 8**

Subtract structure arrays 3 and 8 and store the result in structure 10 with a dataset comment, 'Total Charge - Line + Diode'

**SUBSTR, 3, 8, 10, c= 'Total Charge - Line + Diode'**

## SUM

Calculate the sum or average of multiple WDF arrays.

**Note:** The first WDF array specified determines the maximum domain for the SUM. If a WDF array has valid data but there has no overlap with the domain of the first WDF array specified, then its data will not be reflected in the total but the number of curves which comprise the total will include this WDF array.

**Note:** The last WDF array is used as a work array in calculating the sum. The function *GET\_MAXWDF()* can be used to determine this number.

**format:** SUM, WD1, NCURV [, SAVE] [, /PLOT] [, /AVERAGE] [NCURVES = NCURVES] [, CLABEL = clabel] [, XLABEL = xlabel] [, YLABEL = ylabel]

where:

- wd1* - WDF array number or an array of WDF array numbers
- ncurv* - Number of curves to be totaled or the destination for the sum or average according to the following:  
If *wd1* is a single WDF array number, then *wd1* and the following *ncurv-1* will be totaled. *Ncurv* is a required input parameter in this case.  
If *wd1* is an array of WDF array numbers, *ncurv* will be interpreted as *save*. *Ncurv* is an optional input parameter in this case.
- save* - WDF array number for the sum or average of the WDF arrays. The total, or sum, is always stored in the maximum WDF array (*maxwdfarray*). The average will be stored only in *save*.
- plot* - If present and not zero, plot the sum of the WDF arrays
- average* - If present and not zero, calculate the average by dividing the sum by the number of WDF arrays which have valid data.
- ncurves* - Returned with the number of WDF arrays which were used for the sum. (*sum \* ncurves = average*)
- clabel* - Dataset comment for the sum  
Default is the dataset comment for *wd1*
- xlabel* - X-axis label for the sum  
Default is the x-axis label for *wd1*
- ylabel* - Y-axis label for the sum  
Default is the y-axis label for *wd1*

Examples:

Add arrays 2 and 3 and store the result in array 8. The first example will also store the sum in the last WDF array(given by *get\_maxwdf()*).

```
SUM, 2, 2, 8 <or> add, 2, 3, 8
```

Add arrays 2 through 7 and store the result in array *total*. *Total* must be in the range [1, *get\_maxwdf()* ] or be undefined.

```
SUM, 2, 6, total
```

Add arrays 12 through 28 and store the result in array 8

```
SUM, 12, 28-11, 8
```

Add arrays 2, 4, 5, 6, 12, 15, plot the sum and store the result in array 8

```
SUM, [2, 4, 5, 6, 12, 15], 8, /p
```

Add arrays 2 through 7, calculate the average and store the average in array 7 with the sum in array *get\_maxwdf()* and the number of arrays in *narr*. Plot the arrays and the average. Since the average is stored in one of the arrays, the arrays must be plotted first.

**plo, 2, 6, /ov**

**SUM, 2, 6, 7, /average, n = narr**

**plo, 7, /no, color = 4**

## SWAP

Reverses background and drawing colors for future plots

**Note:** Does not change color table so existing plots are not modified. See *swapct* to modify color table

**format:** **SWAP**

**Technique:**

switches *!p.color* and *!p.background*

## SWAPCT

Reverses background and drawing colors for future plots by exchanging the colors in the color table.

**Note:** Existing plots are modified, but loading a new color table will change everything back to the original colors. See *swap* to modify the drawing colors

**format:** **SWAPCT [, /RESET]**

**where:**

*RESET* - restores original color table

**Technique:**

switches the color table entries for *!p.color* and *!p.background*

## SYMBOL\_LIST

Special symbols available with *MKSYMBOL* and their number and character representation

#	Name	Character
0	Square	SQ
1	Circle	CI or O
2	Triangle	TR
3	Diamond	DI
4	Big X	BI or X
5	Delta	DE
6	Star	ST
7	Plus	PL or +

8	Hexagon	HE
9	Pentagon	PE
10	Clover	CL
11	PACMAN	PA
12	Spiral	SP

## THERMOMETER

Produce a thermometer type legend for a contour plot. Unless the position keyword is used the thermometer will be on the right end or top of the current plot. Internals: The command uses a contour plot with  $X = [0, 1]$  and  $Y = \text{Amplitude}$  with the  $c\_colors$  given by  $col\_values$ , and  $FILL = 1$  (See Procedure below) The values for  $!x$  and  $!y$  variables are saved when this procedure is called and are restored at completion.

**format:** THERMOMETER, amplitude, col\_values, TITLE = title, COLOR= color, POSITION = position, /data, /horizontal, ABSOLUTE = absolute, TXSIZE = txsize, TYSIZE = tysize, TICKVALUE = tickvalue, CUSOR = cursor, BWHITE = bwhite, IGNORE = ignore, BOX = box, BFILL = bfill, CBOX = cbox, THICK = thick, CHARSIZE = charsize, \_EXTRA = extrastuff

where:

- amplitude* - Array of values - typically levels of the contour plot
- col\_values* - Array of color values - typically levels of the contour plot  
**Note:** Amplitude and col\_values must have the same number of values
- Exception* - If amplitude has 3 values and col\_values has 2 and  $amplitude(1) < amplitude(0) < amplitude(2)$  then a two color thermometer will be drawn with col\_values(0) below (or to the left of) amplitude(0) and col\_values(1) above (or to the right of) amplitude(0). See also / bwhite
- BWHITE* - If set, col\_values is set to [ $!p.color$ ,  $!p.background$ ]
- COLOR* - Color for printing.  
 Default is  $!p.color$  (white for X and Black for PostScript) Values are: 1, red; 2, green; 3, blue; 4, magenta; 5, orange; 6, cyan; 7, yellow; 8, white; 0, black
- CURSOR* - If set the two opposite corners of the thermometer will be set with a cursor. Normalized position will be returned in position.
- POSITION* - Position on the plot for the thermometer in units is determined by keywords. If position has valid data, TXsize and TYsize are ignored.

- If POSITION is undefined then the default is determined by the value of txsize and tysize. POSITION will be returned with the values in normal coordinates.
- IGNORE* - Thermometer tries to adjust the title position to fit. If it moves it to a bad location, use ignore. If ignore is not equal to 1, then the position will be moved an amount left or right given by the difference between ignore and 1
- /data* - Determines the coordinates for the position keyword Data refers to the plot coordinates. If keyword is not set, then normalized space coordinates are used.
- Note: Normalized coordinates are relative to the plot not the entire screen, ie, x = 1 will be the middle of the screen or page if !p.multi(1) equals 2
- ABSOLUTE* - If present and not zero then position is normalized coordinates relative to the screen not the last plot
- /HORIZONTAL* - If keyword is set, thermometer will be horizontal. If set, default value of Txsize = [.5, .9] and Tysize = [.3, .6]\*(!y.window(0) - !y.region(0)) + !y.region(0)  
Default is vertical
- TXsize* - If position is not specified, this parameter determines the horizontal size of the thermometer and optionally the positional x values if TXsize\*(!x.region(1) - !x.window(1))  
Default is TXsize = 0.3  
Default position is against the right border.  
If two values are given, these are used as the two positional x parameters
- TYsize* - If position is not specified, this parameter determines the vertical size of the thermometer and optionally the positional y values if TYsize is a two element vector. If a single parameter is given, the height of the thermometer will be given by: TYsize\*(!y.window(1)-!y.window(0))  
Default value is TYsize = 0.6  
Default position is vertically centered.  
If two values are given, these are used as the two positional y parameters.  
Example:  
TXsize and TYsize are two element arrays.  
position = [txsize(0), tysize(0), txsize(1), tysize(1)]
- THICK* - parameter sent to plot, changes the thickness of the

- frame
- CHARSIZE* - parameter sent to plot.  
default is  $1/\max(!p.\text{multi}(1:2) - 1) > 1$
- BOX* - Draw a box around the thermometer is set. If box has 4 elements, they will be taken as the position of the box. Otherwise the size will be estimated. The color will be specified by *CBOX*
- CBOX* - Color of the BOX. Default is *COLOR*
- BFILL* - Fill the box with the specified color. Note to erase an area: *BFILL* = *!p.background*
- Title* - Title for the thermometer. This may be a string array. Each element is printed on separate lines. For a vertical thermometer the title will be centered over an extension of the Y axis unless it will extend past the edge of the plot region; otherwise it will be moved left or right to lie with the plot region. For a horizontal thermometer, the title will be centered over the thermometer.
- Tickvalue* - Value where thermometer values are to be printed. Use */xstyle* for horizontal and */ystyle* for normal thermometers if exact scaling is desired.
- \_EXTRA* - Any valid parameters for the plot command *xstyle* or *ystyle* = 1 will force exact scaling

#### Procedure:

Set up the position if not specified and:

```
plot, amplitude, /nodata, position = position, color = color, _extra =
extrastuff, xtickname = [ ' ', ' ' ], xticks = 1, xticklen = 1e-3
```

Determine the number of numbers and repeat as required the following:

```
contour, data, [0, 1], amplitude, position = position, color = color,
c_color = col_values, levels = amplitude, thick = thick
```

#### Examples:

Add a thermometer to a contour plot of data in structure 23

```
plotstr, 23, / THERMOMETER
```

Add a thermometer to a E field plot from str 2 and scale the data. Take Er and slice the data at theta = 0.0 and plot 25 levels

```
plotstr, 2, 1, 2, 0.0, /THERMOMETER, m =1e-8, nl = 25, ther = {title:['Er Field',
'(MV/cm)]}, title = 'Radial Electric Field'
```

or

```
lev = 0 & color = -1
```

```
plotfld, 2, 1, 2, 0.0, color = color, nl = 25. level = lev, xmargin = [10, 10], title =
'Radial Electric Field'
```

```
THERMOMETER, lev*1e-8, color, title = ['Er Field', '(MV/cm)']
```

Add a thermometer to a plot with a range of [0, 10] and a value of 5.6 and color 1 below 5.6 and color 2 above 5.6; make a two line title of 'Diode Volatge (MV)'; use the cursor to place

**THERMOMETER, [5.6, 0, 10] , [1, 2], tit = ['Diode', 'Voltage (MV)'], /cur**

**Note:** to change to horizontal merely add the “, /horizontal” keyword.

To make a black and white thermometer merely change [1, 2] to [!p.color, !p.background] or set the keyword /bwhite as:

**THERMOMETER, [5.6, 0, 10] , /bw, tit = ['Diode', 'Voltage (MV)'], /cur**

Add a black and white thermometer to a plot which has a value of volt(i) and ranges from 0 to 5

**THERMOMETER, [volt(i), 0, 5], /bw**

## TLPARAM

This procedure computes and (optionally) diplays transimission line parameters as a function of frequency from voltage/current/power time histories.

**format: TLPARAM, Wdf1a, Wdf1b, Defn, Wdf2a, Wdfout, Distance**

where:

First define the impedance using the first 3 parameters: Impedance is  $Z = V/I = V^2/P = P/I^2$ . Any combination of V, I and P may be used and indicated in Defn

- Wdf1a* - voltage, current or power time history at position #1
- Wdf1b* - voltage, current or power time history at positoin #1
- Defn* - definition of impedance: 'iv', 'vi', 'pv', 'vp', 'pi', 'ip' Alternatively, 1, 2, 3, 4, 5, 6 may be used. Where I represents current, V is voltage, and P is power and the two elements are in Wdf1a and Wdf1b respectively Default = 'IV'
- Wdf2a* - voltage, current or power time history at position #2. Must be the same parameter as in Wdf1a
- Wdfout* - WDF array number to store transmission line parameters. Output will be stored in this WDF array and next 6 arrays. (see Outputs)
- Distance* - Distance in meters between Position #1 and Position #2

Keyword Parameters:

- Res* - maximum desired frequency resolution  
Default value is the previous value if this routine has been run in the current session or  $1/(np*dt)$

where  $np$  is the number of points and  $dt$  is the point spacing.

**Fmax** - maximum frequency component to compute  
 Default value is the previous value if this routine has been run in the current session or the value  $\min(1024 \cdot res, 0.5/dt)$  where  $dt$  is the point spacing

**Note:** Res and Fmax are approximate and will be used as guides.  $dt$  will be modified to produce good values for the FFT

**Quiet** - if present and non-zero, the parameters are not displayed

**GHz** - if present and non-zero, use frequency units of GHz

**MHz** - if present and non-zero, use frequency units of MHz

Outputs:

**Wdfout** -  $\text{Re}(Z_0)$  (Real part of the impedance)

**Wdfout+1** -  $\text{Im}(Z_0)$  (Imaginary part of the impedance)

**Wdfout+2** -  $\alpha = \text{Re}(\gamma)$

**Wdfout+3** -  $\beta = \text{Im}(\gamma)$

**Wdfout+4** -  $\epsilon_{\text{eff}} = (\beta \cdot c / \omega)^2$

**Wdfout+5** -  $\text{PUL L} = Z_0 \cdot \beta / \omega$

**Wdfout+6** -  $\text{PUL C} = \beta / \omega \cdot Z_0$

## TOTFR

Add the counts in a CR-39 digitized image. Input structure must be single block, 2 dimensions.

**format:** `Track_Total = TOTFR(STR_IN [, NX] [, NY] [, AREA = area] [, DENSITY = density] [, /PLOTIT])`

where:

**Track\_Total** - Total number of tracks in the designated area. If NX is present and  $> 0$ ,  $NX \cdot NY$  values will be returned. If NX is zero or omitted, Track\_Total is a scalar value.

**STR\_IN** - Structure of structure array number with the CR-39 image. This may

**NX** - Number of images in the horizontal direction. If NX is omitted, zero or negative, then the user will be asked to designate a single area.

**NY** - Number of frames in the vertical direction. Ignored if NX is not positive.  
 Default value = 1

- AREA** - Keyword returned with area =  $S_x * S_y * D_x * D_y$  Where  $S_x$  and  $S_y$  are the size in bins in the X and Y directions of each image and  $D_x$  and  $D_y$  are the size of the bins. AREA is a scalar quantity.
- DENSITY** - Average number of Tracks /unit area.  
=Track\_Total/AREA
- PLOTIT** - If set, Image will be plotted.

**Background:**

CR-39 images are digitized according to the number of tracks with a given size in a bin. Low resolution scans are 300 microns/bin. High resolution scans are 150 microns/bin.

Frame number	Track Area (Pixels = $0.299 \text{ um}^2$ ) / Effective Diam.
1	1 to 2 pixels / 0.617 to 0.872 microns
2	3 to 5 pixels / 1.069 to 1.380 microns
3	6 to 10 pixels / 1.511 to 1.951 microns
4	11 to 16 pixels / 2.046 to 2.468 microns
5	17 to 23 pixels / 2.544 to 2.959 microns
6	24 to 32 pixels / 3.023 to 3.490 microns
7	33 to 42 pixels / 3.544 to 3.999 microns
8	43 to 53 pixels / 4.046 to 4.492 micron
9	54 to 65 pixels / 4.534 to 4.974 microns
10	66 to 94 pixels / 5.013 to 5.982 microns
11	95 to 128 pixels / 6.014 to 6.981 microns
12	129 to 168 pixels / 7.008 to 7.997 microns
13	169 to 212 pixels / 8.021 to 8.984 microns
14	213 to 262 pixels / 9.005 to 9.987 microns
15	263 to 378 pixels / 10.006 to 11.996 microns
16	379 to 514 pixels / 12.012 to 13.989 microns
17	515 to 672 pixels / 14.002 to 15.995 microns
18	673 to 851 pixels / 16.007 to 17.999 microns
19	852 to 9999 pixels / 18.010 to 112.832 microns
20	Saturation Frame - Total Number of Pixels with tracks normalized to the number of pixels in the bin times 32768.

(a value of 16384 means half of the pixels contain tracks)

**Examples:**

Calculate the total number of tracks for the entire structure 1

**tracks = TOTFR (1, 1)**

Calculate the number of tracks in each of 6 images in structure 2 which are arranged

**tracks = TOTFR(1, 3, 2)**

**Note:** tracks = tracks[3, 2]

**TRIFL**

Model electromagnetic energy transport in a transmission line Source voltages are either from a WDF array or analytic Loads are Constant Impedance or perveance, analytic falling impedance, analytic rising then falling, impedance from a wdf array, or an imploding plasma load Electron flow in the MITL is assumed to occur by default. Electron flow can be turned for the entire MITL or up to a point on the transmission line. After flow is turned on, it will be on until the load.

format: TRIFL [, X ,ZVAC] [, Zflow] [, Transition = TR] [, OUT = out]  
[, CONVOLUTE = convolute] [, FLOWOUT = flowout]

format: TRIFL [, WD\_ZV] [, WD\_ZF] [Transition = TR] [, OUT = out]  
[, CONVOLUTE = convolute] [, FLOWOUT = flowout]

format: TRIFL [, X, Y] [Transition = TR] [, OUT = out] [, CONVOLUTE = convolute]  
[, FLOWOUT = flowout]

where:

**Note:** X values cannot be closer than 1e-5 of the largest value. Values closer then this will be averaged. All dimensions are meters/ohms

*X, Zvac, Zflow* - At least 2 arrays must be provided and have the same number of elements where:

X - spatial coordinate

Zvac - Vacuum Impedance

Zflow - Flow impedance (In general, Ignored and calculated according to the formula:  $Z_{flow} = Z_{vac} * v / (v + 2Me)$  where Me is 0.511 kV and v is the peak voltage on the line. At a convolute Zflow is held constant until the convolute region is allowed to transition to the downstream value.

*WD\_Zv, WD\_Zflow* - Waveform array numbers for the vacuum impedance and the flow impedance. (See Zflow above - largely ignored)

**Note:** Linear interpolation will be used to connect input points.

*X, Y* - Arrays from the READASC command  
If Y has 3 values in the second dimension then TR = Y(\*, 2)

X - spatial coordinate

Y(\*, 0) - Zv - Vacuum Impedance

Y(\*, 1) - Zflow - Flow impedance

*Transition* - A single element or an array indicating the type of transition between transmission line segments. If TR is an array it must have the same number of elements as X. If waveform arrays are provided, linear interpolation will be used to put the waveforms on a common time scale.

Values are:

0 Linear (Default)

1 Sine<sup>2</sup>

2 Log

*OUT* - Starting point for WDF arrays and Structure Arrays  
Default is *OUT* = 1

*CONVOLUTE* - An array of values corresponding to the convolute flow impedance; a zero value indicates no convolute

*FLOWOUT* - Flow impedance on the output of a convolute; a zero value indicates no convolute

**Note:** The environment variable "TRIFL\_EXE" will be used as the executable if it is specified

Outputs:

Three file are written:

1. Suffix of ".geom" is an ASCII geometry file for the TRIFL
2. Suffix of ".src" is an ASCII source file for the TRIFL code
3. Suffix of ".pff" is the calculated data for the TRIFL run.

The first dataset is an IFL dataset.

The integer array has the load type and the electron flow flag

The float list has the source impedance for the driver and in the case of a Imploding Plasma Load, it had the following: Three file are written:

Mass/Length, Length, Initial Radius, /% to FinalRadius, Return Radius.  
L0, Lstart, I0

The following WDF arrays are populated for *OUT*=1 and a Imploding Plasma Load:

Array No.	Dataset Comment
1	Load Voltage vs Time
2	Load Current vs Time
3	Input Voltage vs Time
4	Input Current vs Time
5	dL/dt - Change in Inductance vs Time
6	Inductance vs Time
7	Forward Going Voltage
8	Forward Going Power vs Time
9	Forward Going Energy vs Time
10	Input Energy vs Time
11	Output Energy vs Time
12	Inductive Load Energy vs Time
13	Net Foil Energy vs Time
14	Total Efficiency (wdf 13/wdf9 * 100)

For a resistive load the arrays are:

Array No.	Dataset Comment
1	Load Voltage vs Time
2	Load Current vs Time
3	Input Voltage vs Time
4	Input Current vs Time
5	Forward Going Voltage
6	Forward Going Power vs Time
7	Forward Going Energy vs Time
8	Input Energy vs Time
9	Output Energy vs Time
10	Total Efficiency (wdf 9/wdf7 * 100)

The following structure arrays are populated for OUT = 1

Array No.	Dataset Comment
1	A -
2	V -
3	Q -
4	Ie -
5	Ia -
6	Ic -
7	Qa -
8	Qc -

#### COMMON BLOCKS:

The transmission line data is stored in a common block: MITL\_common

#### PROCEDURE:

Technique developed by C. W. Mendel with some modifications by L. P. Mix and D. B. Seidel.

Each transition line section will need the vacuum impedance as a function of axial impedance.

#### EXAMPLE:

TRIFL

## TRIFLW

Model electromagnetic energy transport in a transmission line Source voltages are either from a WDF array or analytic Loads are Constant Impedance or perveance, analytic falling impedance, analytic rising then falling, impedance from a wdf array, or an imploding plasma load Electron flow in the MITL is assumed to occur by default. Electron flow can be turned for the entire MITL or up to a point on the transmission line. After flow is turned on, it will be on until the load.

format: TRIFLW [, X ,ZVAC] [, Zflow] [, Transition = TR] [, OUT = out]  
 [,CONVOLUTE = convolute] [, FLOWOUT = flowout]

format: TRIFLW [, WD\_ZV] [, WD\_ZF] [Transition = TR] [, OUT = out]  
 [,CONVOLUTE = convolute] [, FLOWOUT = flowout]

format: TRIFLW [, X, Y] [Transition = TR] [, OUT = out] [,CONVOLUTE = convolute]  
 [, FLOWOUT = flowout]

where:

**Note:** X values cannot be closer than 1e-5 of the largest value. Values closer then this will be averaged. All dimensions are meters/ohms

*X, Zvac, Zflow* - At least 2 arrays must be provided and have the same number of elements where:

X - spatial coordinate

Zvac - Vacuum Impedance

Zflow - Flow impedance (In general, Ignored and calculated according to the formula:  $Zflow = Zvac * v / (v + 2Me)$  where Me is 0.511 kV and v is the peak voltage on the line. At a convolute Zflow is held constant until the convolute region is allowed to transition to the downstream value.

*WD\_Zv, WD\_Zflow* - Waveform array numbers for the vacuum impedance and the flow impedance. (See Zflow above - largely ignored)

**Note:** Linear interpolation will be used to connect input points.

*X, Y* - Arrays from the READASC command  
 If Y has 3 values in the second dimension then TR = Y(\*, 2)

X - spatial coordinate

Y(\*, 0) - Zv - Vacuum Impedance

Y(\*, 1) - Zflow - Flow impedance

*Transition* - A single element or an array indicating the type of transition between transmission line segments. If TR is an array it must have the same number of elements as X. If waveform arrays are provided, linear interpolation will be used to put the waveforms on a common time scale.

Values are:

0 Linear (Default)

1 Sine<sup>2</sup>

2 Log

*OUT* - Starting point for WDF arrays and Structure Arrays  
 Default is OUT = 1

*CONVOLUTE* - An array of values corresponding to the convolute flow impedance; a zero value indicates no

*FLOWOUT* convolute  
 - Flow impedance on the output of a convolute; a zero value indicates no convolute  
**Note:** The environment variable "TRIFL\_EXE" will be used as the executable if it is specified

## Outputs:

Three file are written:

1. Suffix of ".geom" is an ASCII geometry file for the TRIFL
2. Suffix of ".src" is an ASCII source file for the TRIFL code
3. Suffix of ".pff" is the calculated data for the TRIFL run.

The first dataset is an IFL dataset.

The integer array has the load type and the electron flow flag

The float list has the source impedance for the driver and in the case of a Imploding Plasma Load, it had the following: Three file are written:

Mass/Length, Length, Initial Radius, /% to FinalRadius, Return Radius.  
 L0, Lstart, I0

The following WDF arrays are populated for OUT=1 and a Imploding Plasma Load:

Array No.	Dataset Comment
1	Load Voltage vs Time
2	Load Current vs Time
3	Input Voltage vs Time
4	Input Current vs Time
5	dL/dt - Change in Inductance vs Time
6	Inductance vs Time
7	Forward Going Voltage
8	Forward Going Power vs Time
9	Forward Going Energy vs Time
10	Input Energy vs Time
11	Output Energy vs Time
12	Inductive Load Energy vs Time
13	Net Foil Energy vs Time
14	Total Efficiency (wdf 13/wdf9 * 100)

For a resistive load the arrays are:

Array No.	Dataset Comment
1	Load Voltage vs Time
2	Load Current vs Time
3	Input Voltage vs Time
4	Input Current vs Time
5	Forward Going Voltage

6	Forward Going Power vs Time
7	Forward Going Energy vs Time
8	Input Energy vs Time
9	Output Energy vs Time
10	Total Efficiency (wdf 9/wdf7 * 100)

The following structure arrays are populated for OUT = 1

Array No.	Dataset Comment
1	A -
2	V -
3	Q -
4	Ie -
5	Ia -
6	Ic -
7	Qa -
8	Qc -

#### COMMON BLOCKS:

The transmission line data is stored in a common block: MITL\_common

#### PROCEDURE:

Technique developed by C. W. Mendel with some modifications by L. P. Mix and D. B. Seidel.

Each transition line section will need the vacuum impedance as a function of axial impedance.

#### EXAMPLE:

TRIFLW

## TSH

Performs time shift on a WDF array. This procedure can be used to synchronize two arrays or to insure an array begins at zero.

**format:** TSH, WD1 [, AMOUNT]

where:

- |               |   |
|---------------|---|
| <i>wd1</i>    | - WDF array number  |
| <i>amount</i> | - time shift. If <i>amount</i> is zero or omitted, then the array will start at zero by deleting all points before zero or by adding zero amplitude points if <i>wd1</i> begins after zero. |

#### Examples:

Make array 1 begin at an abscissa value of zero. Add initial zeros if necessary.

TSH, 1

Time shift (add to the abscissa values of) array 1 the amount,  $-2.42e-6$   
**TSH, 1,  $-2.42e-6$**

## USERQSINIT

Initialize default conductor/grad parameters and the plot orientation

**format:** This routine is called automatically by the QS/PFIDL procedures

Each user can put his own version of this routine in his own procedure library and remove the comment character from the lines he wishes. The present version sets the following parameters.

use: cp /usr/local/lib/idl/lib/contrib/userqsinit.pro {local directory}

<i>transdefault</i>	- interchange x and y axis in field plots load default color tables default values are loadct, 5 loadxyct
<i>concolor</i>	- store the default color for conductors
<i>conthick</i>	- store the default thickness for conductors
<i>grdcolor</i>	- store the default color for grids
<i>grdthick</i>	- store the default thickness for grids
<i>fldcolor</i>	- store the default color for field plots (PLOTFLD)
<i>directorypage</i>	- store the default page size for scroll directory (DIRP)
<i>directoryhelp</i>	- store the default for directory help window
<i>PATH</i>	-Initial path for <i>PICKFILE()</i>
<i>FILTER</i>	-Initial filter for <i>PICKFILE()</i>

## VIDA2STR

Convert a vida image to a PFIDL structure.

**format:** VIDA2STR, vida, str [, /structure] [, CLABEL = clabel]  
 [, DX = dx, DY = dy, DZ = dz]

where:

<i>vida</i>	- Vida data type 1 = Film Intensity 2 = Film Density.
<i>str</i>	- Variable name or a structure array number
<i>structure</i>	- If set then data is returned in STR. If not set then data is stored in a stucture.
<i>CLABEL</i>	- Label used for dataset comment. Default is Title_1
<i>DX, DY, DZ</i>	- Axis spacing for each dimension

EXAMPLE:

Make a sturcture array from the current intensity array in vida and store in picture; use dataset description 'Filtered Image'

**VIDA2STR, 1, picture, clab = 'Filtered Image', /structure**

Store in structure array, 3, the current density image in vida

**VIDA2STR, 2, 3**

## VIDA\_AXIS

Modify the axis scaling of vida data. Vida stores offsets in mm and dx in microns. This routine takes the initail point and changes the dx by a factor of 1000.

**format: VIDA\_AXIS, sTR1 [, STR2]**

where:

*STR1, STR2* - Structure array numbers.  
Data in STR1 is converted and stored in STR2. If STR2 is missing, then data is stored in STR1. Data must be type 1 field data

**EXAMPLE:**

Fix data in structure 4 and return to structure 4

**VIDA\_AXIS, 4**

Fix data in structure 5 and return to structure 6

**VIDA\_AXIS, 5, 6**

## W132

Set screen width to 132 characters

**format: W132**

**Note:** Must be in an xterm with 80/132 switching enabled

Examples:

Set screen to 132 characters.

**W132**

## W2I

Converts a WDF array to IDL arrays or to an IDL structure

**format: W2I, XARRAY, YARRAY, WD1 [, CLABEL = clabel] [, XLABEL = xlabel] [, YLABEL = ylabel] [, FILE = FILE]**

**format: W2I, STR, WD1, /STRUCTURE**

where:

*xarray, yarray* - IDL arrays to be generated from a WDF array.  
*str* - Structure to be generated from a WDF array  
*wd1* - WDF array number  
*clabel* - dataset comment (used for title of plots)

- xlabel* - x-axis (abscissa) label
- ylabel* - y-axis (ordinate) label
- file* - file or source of these dataset values
- structure* - If set, all other keywords are ignored.  
If the number of points is zero, then zeros will be returned for all numbers and null strings for the labels. No error will be sent unless WD1 is out of range.

Structure labels are:

- type* - 6 type for WDF arrays
- ndim* - 1 dimension
- nblk* - number of blocks
- loc* - pointer array to the first element in each block
- range* - 2 element array with the minimum and maximum abscissa values
- yrange* - 2 element array with the minimum and maximum ordinate values
- np* - number of points; zero indicates no data in this array
- x0* - Initial point
- dx* - Spacing for uniform data; -1 for x-y data
- x* - Abscissa values
- y* - Ordinate values
- xlab* - Abscissa label
- ylab* - Ordinate label
- file* - File from which the data originated
- clab* - Dataset comment label

Example:

Convert WDF array 2 to IDL arrays, x and y.

```
W2I, x, y, 2
```

Convert WDF array 2, to IDL arrays, x and y, and store the title and abscissa labels in variables, *title* and *xlab*

```
W2I, x, y, 2, c=TITLE, x= XLAB
```

## W80

Set screen width to 80 characters

format: **W80**

**Note:** Must be in an xterm with 80/132 switching enabled

## Examples:

Set screen to 80 characters.

**W80**

**WDCOMMENT**

This function returns the WDF comment for an array.

**format:** com = WDCOMMENT(WDF)]

**where:**

*WDF* - Valid WDF comment

Example:

Print the comment label for array 5

Print, WDCOMMENT(5)

Add the shot number to a comment of array 3, where shot number is SHOT.

cha, 3, c=WDCOMMENT(3)+' , Shot' +strtrim(shot, 2)

**WDEDIT**

Edit a WDF array shape by adding or deleting points. A reference dataset can be plotted over the curve being edited. The editing is interactive using the cursor.

**Note:** If *wd2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**format:** WDEDIT, wd1 [, wd2] [, OVER = wd3]

**where:**

*wd1, wd2* - WDF array numbers  
if *wd2* is present, the final curve is stored in *wd2*  
otherwise the result is stored in *wd1*

*wd3* - WDF array number of the curve to be plotted over *wd1*

Examples:

Edit data points in array 2 and return the data to array 2

**WDEDIT, 2**

Edit data points in array 2 and return the data to array 3; use array 1 as a reference

**WDEDIT, 2, 3, over = 1**

**WDERR**

Plot error bars over a previously drawn plot

**Note:** get\_maxwdf() is used as a work array

**format:** WDERR, WDdata, relative = rel\_err

**format:** WDERR, WDdata, WDError

**format:** WDERR, WDdata, WDLow, WDHigh

where:

- WDdata* - WDF array number of the data curve
- WDerror* - WDF array number of the error curve  
Error bars extend from (WDdata - WDerror) to WDdata+WDerror)
- WDLow* - Negative error estimate for the data
- WDHigh* - Positive error estimate for the data  
Error bars extend from (WDdata-WDLow to WDdata+WDHigh)

Keywords Parameters:

- WIDTH* - WDF array number of the data curve
  - NPOINTS* - Number of error bars. Default is 18 (see DELTA)
  - DELTA* - Spacing of the error bars. If npoints is greater than zero, then delta will be calculated from NPOINTS. If NPOINTS is undefined or  $\leq 0$  then DELTA will be used as the spacing. If DELTA is larger than the range of the data then it is set to 0. If DELTA is undefined or 0.0 then 18 error bars will be generated. **Note:** DELTA is secondary to NPOINTS. If NPOINTS is provided, then DELTA is ignored.
  - RELATIVE* - If set, then WDerror, WDLow andWDHigh are considered to be the relative error.  
Error bars extend from (WDdata\*(1.0-WDerror)) to (WDdata\*(1.0+WDerror))
- or
- Error bars extend from (WDdata\*(1.0-WDLow)) to (WDdata\*(1.0+WDHigh))
  - If RELATIVE is between 0 and 1 and there is only 1 parameter, then a constant error will be used.
  - SETDEFAULT* - If set, values of NPOINTS and WIDTH will be saved. **Note:** Save value of WIDTH = abs(WIDTH)  
**Note:** Save value of NPOINTS = abs(NPOINTS) > 1
  - SHOWDEFAULT* - If set, show default values of NPOINTS and WIDTH
  - \_EXTRA* - Any valid keyword in call to OPLOT

Side Effects:

An overplot is produced. Last WDF array is used as a work array.

Examples:

To plot symmetrical error bars where for WDF array 3 using error estimates in WDF array 4.

**pl**, 3 ; plot the data  
**WDERR**, 3, 4 ; overplot the error bars

## WDFIT

Fit a WDF array to a polynomial using the IDL contributed function, **svdfit**. The polynomial is of the form:

$$Y = \sum^{degree} coef(i) \cdot X^i ,$$

where *degree* is specified by the user.

**Note:** If *wd2* is present but undefined, then it will be set to the value of the lowest unused WDF array. If all arrays have data, then it will be set to *wd1*.

**format:** WDFIT, wd1 [, wd2] [, DEGREE = degree] [, WEIGHT = weight]  
 [, COEF = coef]

where:

- wd1*, *wd2* - WDF array numbers  
if *wd2* is present, the fitted curve is stored in *wd2*  
otherwise the result is stored in *wd1*
- degree* - degree of the polynomial fit in the equation above  
Default is 1 (linear fit)
- weight* - optional weight array for the points  
Default is 1 (equal weight)
- coef* - polynomial coefficients in the equation above

Examples:

Fit the data points in array 2 to a straight line and return the data to array 2

**WDFIT, 2**

Fit data points in array 2 and return the polynomial data to array 3; use a second order polynomial for the fit.

**WDFIT, 2, 3, degree = 2**

## WDFIT3

Fit a WDF array to a polynomial using the IDL user function, **SVDFIT**. Up to 3 different polynomials may be used. Calculated fit is stored in a WDF array and , optionally, written to a file.

**Note:** *Scale\_regrid*, with default parameters, is used to calculate values.

**format:** WDFIT3, wd1 [, wd2] [, /YLOG] [, /YTYPE] [, VALID = valid] [, FUDGE = fudge]  
 [, /Exact\_Range]

where:

- wd1*, *wd2* - WDF array numbers. If *wd2* is present, the final curve is stored in *wd2* otherwise the result is stored in *MAXWDFARRAY*.

- YLOG* - Same as ytype
- YTYPE* - If present and not zero, then a log fit in the Y direction is performed. Only points greater than zero will be used in the log direction.
- VALID* - Domain of validity for the fit  
Default: valid = [0.95\*minimum, 1.02\*maximum]
- Exact\_Range* - If set VALID = [minimum, maximum]
- FUDGE* - 2 element vector sent to scale\_regrid (see PFIDL2/scale\_regrid). Used to insure that the fitted function is continuous.  
Default values are .05 of VALID

**Discussion:**

The first execution of the command will be performed with:

Number of Breaks = 2

Break Points = [Minimum + range/16, Maximum - range/16] where range = maximum - minimum

Degree = [2, 5, 3]

In the case of a singular value degree will automatically be reduced to obtain a fit. Current version adds a minimum of one point past the break point and will add two points if it is within 1/16 of the domain of the fit.

**Example:**

Fit data point in array 2 to a polynomial multiple polynomials. Save the fit in array 3.

**WDFIT3, 2, 3**

## WDSLPLIT

Split a waveform into two parts. Either an exponential or a linear ramp can be selected. The start point and the

- a. Zero point for a linear ramp.
- b. Exp(-3) point for an exponential

is selected. The remainder is the difference between the original waveform and the first part.

**format:** WDSPLIT, WDF [, \_EXTRA = extrastuff]  
or WDSPLIT, WDF, WDF2 [, \_EXTRA = extrastuff]  
or WDSPLIT, WDF, WDF1, WDF2 [, \_EXTRA = extrastuff]

*where:*

- WDF* - Waveform array which is to be split
- WDF1* - Storage location for First Half of WDF  
Default location is to replace WDF
- WDF2* - Storage location for Second Half of WDF  
Default location is WDF+1

**Keywords:**

- X/YSIZE* - Size of the graphics window
- LCTYP* - Type of fit  
0 = Linear

1 = Exponential

Any keywords use by the PLO command

**NOTE:** Mouse buttons are mapped to widget buttons, cursor must be in the DRAW area of the main widget for the following mouse commands

- Left Button* - Set value  
**Note:** A carriage return in the data field will also set the value.
- Middle Button* - Accpet/Save (current designated lineout will be saved)
- Right Button* - Reset all points ( All points will be undesignedated)  
**Note:** If the last point required to define a curve is selected more than once, the previous “last point” is replaced.
- Set Point Buttons* - These perform the same function as a <CR> in either the X or Y windows above the buttons.

Example:

Split array 60 and put the parts in 60 and 61

**WDSPLIT, 60**

Split array 40 and put the parts in 40 and 50

**WDSPLIT, 40, 50**

Split array 45 and put the first and second halves in 50 and 60. Do not change 45 andplot only the range [30, 90]

**WDSPLIT, 45, 50, 60, xrange = [30, 90]**

## WDVALID

This function returns an array of valid WDF array numbers. Minimum , maximum and a search string can be specified.

**format:** `array = WDVALID([string] [, Maximum = maximum] [,Mimumum = minimum] [,Npoint = npoint] [,case_sensitive = case_sensitive])`

where:

- array* - WDF array numbers which meet the specified criteria
- string* - search string. Leading and trailing blanks are ignored. String may be a number. This routine uses:  
`str = strtrim(string(0), 2)`  
where *str* is the actual search string  
If *string* is missing, all valid arrays are returned.
- minimum* - Minimum value desired, Default = 1
- maximum* - maximum value desired, Default = maxwdfarray
- npoint* - number of values returned
- case\_sensitive* - if set makes the search case sensitive

**Caution:** CASE is a special IDL word so use /c or /ca or /cas or /case\_ or more letters but /case will give an error.

**Note:** If NPOINT = 0, ARRAY = -1

Examples:

Find arrays containing data and store in VALID

```
valid = WDVALID)
```

Find arrays with "bdot" and average them and store the result in 34

```
sum, WDVALID('bdot'), 34, /average, clabel = 'Average bdots'
```

Same as above but a more complex example.

```
arr = WDVALID('bdot', n = n)
```

```
print, 'Set the baseline'
```

```
for i = 0, n-1 do begin & xcurw, arr(i), x, y & sub, arr(i), 0, y(0) & end
```

```
comment = 'Average of bdots:'
```

```
for i = 0, n-1 do comment = comment + ' ' +strtrim(wdcomment(arr(i)), 2)
```

```
sum, arr, 34, /average, clabel = comment
```

## WHATS\_NEW

Print new features on IDL and PFIDL

```
format: WHATS_NEW
```

## WHELP

Lists the dataset comments for each of the WDF arrays containing valid data.

```
format: WHELP [, WDA_1] [, WDA_2] [,NPOINT = npoint] [ /FULL]
```

where:

- |                     |  |
|---------------------|--|
| <i>wda_1, wda_2</i> | - indicate the first and last waveform datasets to be listed. If missing, all datasets are listed.   |
| <i>NPOINT</i>       | - Indicates the number of points to be printed with a full listing. Default is 0. If NPOINT is less than zero, all points will be listed. If NPOINT is greater than zero, NPOINT points will be listed. If NPOINT is specified, then FULL = 1. |
| <i>FULL</i>         | - Indicates the full array information is to be printed if present and not zero. If NPOINT is specified, then FULL = 1   |

Note: If *wda\_2* is missing, default is FULL = 1

If *wda\_2* is present, default is FULL = 0

Print a full listing of the WDF array values for arrays 1 thru 3

```
WHELP, [1, 2, 3]
```

```
(or) WHELP, 1, 3, /fu
```

```
(or) WHELP, 1, 3, n=0 (prints no data values only the range of data)
```

**Note:** If the data was put into a WDF array using IDL2WDF (I2W), then no file will be stored with the array.

Examples:

Print a directory of the dataset comments for all WDF arrays with data

**WHELP**

Print a directory of the dataset comments for datasets 20 to 30 of the WDF arrays and the file from which it was read.

**WHELP, /fi**

## WIN

Limit the extent of a dataset. The cursor is used to indicate the desired initial and final values of the abscissa; therefore, a curve with relevant abscissa values must be plotted before a window command can be issued. This command is similar to the LIM command, except the cursor is used to designate the domain of the array.

**format:** WIN, WD1 [, START] [,STOP]

where:

<i>wd1</i>	- WDF array number
<i>start</i>	- initial abscissa value
<i>stop</i>	- final abscissa value

Example:

Window array 1 using the cursor.

**WIN, 1**

Window arrays 2 through 10 to the same domain (requires two commands.)

**WIN, 2, beg, last**

**lim, 3+indgen(8), beg, last**

## WRI

This procedure writes a WDF array to a PFF file.

**format:** WRI, WD1 [, FILEID = FID] [, CLABEL = CLABEL] [, YLABEL = YLABEL]  
[, XLABEL = XLABEL]

where:

<i>wd1</i>	- WDF array number
<i>fileid</i>	- PFF file id. If <i>fileid</i> is missing or 0, the current PFF file is used.
<i>clabel</i>	- dataset comment. If not specified, the WDF comment is used.
<i>xlabel</i>	- dataset x-axis label. If not specified, the WDF x-axis label is used.

*ylabel* - dataset block label (y-axis label). If not specified, then the y-axis label is used as the block label.

#### Examples:

Write WDF array 1 to the current PFF file.

**WRI, 1**

Write WDF array 3 to the PFF file with a file id of 4.

**WRI, 3, f = 4**

Write WDF array 1 to the PFF file with a file id of 1 and use the dataset comments and block labels indicated.

**WRI, 1, f= 1,c='Calculated Diode Power, Shot 4443', y='Power-W'**

## WRIASC

Write a number of WDF arrays to an ascii file. File format is determined by the user. The arrays are limited to the region of overlap and put on a common time base using the minimum point spacing of the arrays.

**format: WRIASC, fname, wdfa[, narray] [, formatout=format]**

where:

*fname* - the file name  
If *fname* = 0 then *fname* = 'PFIDLdefault.dat'

*wdfa* - First WDF array or an array of WDF arrays. WDF must be either a single value or an array of narray values.

*narray* - the number of arrays to be written to the file  
If present and greater than 0, then WDF(0) and the next *narray*-1 datasets will be written to a file

*formatout* - output format of the data  
0 - only data values in the file  
1 - number of points followed by data values  
2 - Narray labels followed by data values  
3 - (DEFAULT) number of points followed by narray labels then data values

#### Examples:

Write 5 arrays into file 'PFIDLdefault.dat' from WDF arrays

**WRIASC, 0, 2, 5**

or

**WRIASC, 0, [2,3,4,5,6]**

Write 3 arrays to file, 'testdata' from WDF arrays 3, 6, 9. Assume data format to have no header but only 4 columns of data.

**WRIASC , 'testdata', [3,6,9], form = 0**

## WRIIFL

Write an integer float list to a PFF file.

See also WriteIFL in PFIDL

**format:** WRIIFL, *iarray* [, *flist*] [, *FARRAY*] [, *TLABEL* = *tlabel*] [, *CLABEL* = *clabel*] [, *AP* = *apptype*] [, *FILEID* = *fileid*]

where:

- iarray* - integer array to be written
  - flist* - float list to be written
  - FARRAY* - floating array
  - TLABEL* - dataset type label - the first 16 characters are stored in the directory
  - CLABEL* - dataset comment - the first 64 characters are stored in the directory
  - AP* - application dataset type (User parameter) (default = -3)
  - FILEID* - file id. If it is omitted or zero, the current file is used. If present, the current file pointer will be set to this file.
- Note:** Text strings are limited to 240 characters for the write command.

EXAMPLE:

Write the integers representing the load and source type, and the float parameters [a, b, c, d, e, f] as an ifl dataset

WRIIFL, [loadtyp, sourctyp], [a, b, c, d, e, f], t = 'Run Parameters', c= 'Float List: t0, z0, Vmax, ap =-3

## WRIMSD

Write a structure array to an ascii file. This procedure limited to single block, single vector field data. All labels are limited to 80 characters.

**format:** WRIMSD, *fname* [, *STR*] [, *VIDA* = *vida*] [, *CLABEL* = *clabel*]

where:

- fname* - the file name. If *fname* = 0 then *fname* = 'PFIDLdefault.dat'.
- STR* - STR array of STR array number. If STR is not provided, then keyword VIDA must be set.
- VIDA* - Keyword indicating that VIDA data is to be put into a structure. See VIDA2STR for values. The following command is called:  
VIDA2STR, VIDA, STR, /STRUCTURE
- CLABEL* - Comment label for the output

## EXAMPLE:

Write STR array 5 into file 'PFIDLdefault.dat'  
**VIDA2STR, 1, picture, clab = 'Filtered Image', /structure**  
 Store in structure array, 3, the current density image in vida  
**VIDA2STR, 2, 3**

**WRISTR**

**Caution:** This routine has few seat belts or airbags. It might work but don't be surprised if it doesn't.

For ASCII Write of an image see WRIMSD

## Limitations:

Current version does not specifically support 3D uniform data. This type data will be output as nonuniform data.

Only: Particle, Field and Vector data are supported. Default dataset type for single block data is NGD. NGD limited to 4 spatial dimensions and 6 vector dimensions but this is easily changed.

**format:** WRISTR, STR [, FID] [, CLABEL = clabel] [, TLABEL = tlabel] [, FILEID = fid]  
 [, APTYPE = aptype]

where:

- |                |   |
|----------------|---|
| <i>STR</i>     | - structure array number or a structure   |
| <i>FID</i>     | - file id of pff file. Default is fid = 0 -- ie the current file<br><b>Note:</b> if both FID and FILEID are specified, FID is used.   |
| <i>TLABEL</i>  | - Type-label for dataset. Default is TLAB is used   |
| <i>CLABEL</i>  | - Comment label for dataset. Default is the comment label: CLAB   |
| <i>APTTYPE</i> | - application dataset type:<br><b>Caution:</b> If a normal type is used, when the data is read the energy calculation might mess things up for particle!. Default values are 15 and 27 for fields and particles.<br>Only the first 60 characters are used for labels. |
| <i>NF3</i>     | - If set the data will be written as NF3  |

## EXAMPLE:

Write structure 34 to the current file  
**WRISTR, 34**  
 Write structure F to the file with file id = 3  
**WRISTR, F, fid = 3**  
 or  
**WRISTR, f, 3**

## WRIVTX

Reads a field or charge density into an IDL structure. Procedure options are explained below, Dataset must be type 5, PFFVTX

format: **WRIVTX** [, **ATTRIBUTES**] [, **FILEID** = fileid] [, **XARRAY** = xarray]  
 [, **YARRAY** = yarray] [, **ZARRAY** = zarray] [, **TARRAY** = tarray]  
 [, **APTYPE** = aptype] [, **ALABEL** = alabel] [, **CLABEL** = clabel] [, **TLABEL** = tlabel]  
 [, **XLABEL** = xlabel] [, **YLABEL** = ylabel] [, **ZLABEL** = zlabel] [, **X4Label** = x4label]  
 [, **SPARE** = spare]

where:

- ATTRIBUTES* - IDL Attribute Array size = (NVERT, NATTR)  
 If dataset has no attributes, enter nothing or zero. If valid, the first dimension should have the same size as Xarray.
- FILEID* - file id of the PFF file containing the data
- stringlen* - Maximum number of characters in the spatial labels and the field labels. Maximum of 800 characters total and 64 for each block. (maximum = natr\*stringlen < 800)  
 Default value is 32 characters for each label.

**Note:** The following 4 spatial arrays must have the same number of elements or they will be assumed to not contain valid data. The number of elements in XARRAY is taken as NVERT. The number of spatial arrays with the number of elements equal to NVERT is taken as NDIM (the number of dimensions)

- CLABEL* - Label used for dataset comment.  
 Default is Title\_1
- DX, DY, DZ* - Axis spacing for each dimension

EXAMPLE:

Make a structure array from the current intensity array in vida and store in picture; use dataset description 'Filtered Image'

**VIDA2STR, 1, picture, clab = 'Filtered Image', /structure**

Store in structure array, 3, the current density image in vida

**VIDA2STR, 2, 3**

## WRITEIFL

Write an integer float list to a PFF file. See also WriIFL in PFIDL2

format: **WRITEIFL**, fileid, iarray, flist [, **FARRAY**] [, **TLABEL** = tlabel] [, **CLABEL** = clabel]  
 [, **AP** = aptype]

where:

- fileid* - file id. If it is omitted or zero, the current file is used.

- If present, the current file pointer will be set to this file.
- iarray* - integer array to be written
  - flist* - float list to be written
  - FARRAY* - floating array
  - TLABEL* - dataset type label - the first 16 characters are stored in the directory
  - CLABEL* - dataset comment - first 64 characters are stored in directory
  - AP* - application dataset type (USER parameter) (default = -3)
- Note:** Text strings are limited to 240 characters for the write command.

**Example:**

Write the integers representing the load and source type, and the float type parameters [a, b, c, d, e, f] as an ifl dataset.

```
WRITEIFL, 0, [loadtyp, courtyp], [a,b,c,d,e,f], t = 'Run Parameters', c = 'Float list:
t0, z0, Vmax, Tmax, alp, c', ap = -3
```

**WRITEPFF**

Write an IDL array to a PFF file. Only 1-D, 2-D, and 3-D data may be written in the present version of the interface. The variables are the same as for the read command. Nonuniform 1-D data and all 2-D data will be written as 3-D data at the present time. No interactive input is supported. Datasets are added sequentially to the end of the file.

```
format: WRITEPFF , fileid , image , [ space ] [, TLABEL = tlabel] [, CLABEL = clabel]
[, BLABEL = blabel] [, XLABEL = xlabel] [, YLABEL = ylabel]
[, ZLABEL = zlabel] [, LABEL = label] [, AP = apptype] [, SPARE = ispare]
```

where:

- fileid* - file id. If it is omitted or zero, the current file is used. If present, the current file pointer will be set to this file.
- image, space* - arrays containing the dataset information defined in Table 2 (see READPFF).  
If “space” is not provided, a default grid, beginning at zero (0.0) with a spacing of 1.0 units is used for each dimension.
- tlabel* - dataset type label - the first 16 characters are stored in the directory
- clabel* - dataset comment - the first 64 characters are stored in the directory
- blabel* - title or label for this block
- xlabel* - first dimension label for this block

- ylabel* - second dimension label for this block
- zlabel* - third dimension label for this block
- label* - array of all the above labels in the order given (*label(0) = tlabel, label(1) = clabel, etc.*)  
If "LABEL" is provided, all other label keywords will be appended to the front of the string in the "LABEL" array.
- ap* - application dataset type (User parameter)
- spare* - spare integer array for the application (User specified integer *iarray(5)*)

**Note:** Text strings are limited to 240 characters for the write command.

Examples:

Write an IDL array, *picture*, to the end of the current file. Since no spatial array is provided, a default grid, beginning at zero, with an increment of 1.0 will be provided for each dimension of "picture".

```
WRITEPFF, 0, picture
```

Write an IDL array, *imag*, with spatial array, *grid*, to the end of the file having a file id of 23. Use the strings provided for the various labels.

```
WRITEPFF, 23, imag, grid, t='k-alpha image', c = 'Unfiltered image in quadrant 1',  
$ x= 'Distance - Microns', y= 'Distance - Microns', b= 'Film Density'
```

Write the IDL array, *image*, with spatial array, *space*, to the end of the current file. Use the character array, *lab*, for the labels and comments. The clabel string will be appended to the beginning of the contents of *lab(1)*.

```
WRITEPFF , 0, image, space, l= lab, c= 'Filtered image & subtracted bkgrd of  
0.233'
```

## WRITETK

Write a tkdas data file. Format consists of tab delimited columns of the form:

**Note:** Each Title may not have spaces in this version.

title

t0

delta\_t

Npoints

y1 (at t0)

y2

y3

.

.

.

yN (at t0+(N-1)\*delta\_t)

```
format: WRITETK , fname, WDF [, narray]
```

where:

*fname* - the file name  
 if a file name then the file is opened and closed at the end. If *fname* is zero or omitted then PICKFILE will be used to get the file name

**Note:** If only one parameter is given and the parameter is a number, then READTEK2 assumes that it is a STR number.

*WDF* - WDF array number or an array of numbers  
 Default is 1  
 If *WDF* is less than 0 and *n\_elements(data) > 0* then no WDF arrays will be stored.

*narray* - If *WDF* is a single number then *narray* total arrays will be written beginning with *WDF*

Examples:

Write a TKDAS file with arrays [2, 3, 4, 5, 6] to data.dta

WRITETK, 'data.dat', 2, 5

or

WRITETK, 'data.dat', [2, 3, 4, 5, 6]

## X2PS

Switch to a PostScript file for graphics output. Line thicknesses are reset to the value set when the PostScript file was opened. Startpost must be used to initially open the file.

format: X2PS

## XCUR

Select values at a finite number of points in a window. Procedure will continuously display the (x,y) location of the cursor and store the values when the mouse button is depressed. The cursor can be set at a specific location by editing the displayed values of X and Y and hitting a carriage return in the X or Y display window.

**Note:** Data comes from current plot window. See XCURW to take points from a waveform array.

format: XCUR [ , X , Y] [ , dx, dy] [ , /SORT] [ , /NOSYMBOL]

where:

*x, y* - abscissa and ordinate values when the mouse button is depressed. By default, a symbol will be placed on the plot at these values.

*dx, dy* - change in the abscissa and ordinate values  
**Note:** If data is sorted then *dx* and *dy* will be recalculated.

- sort* - If present and not zero, then data will be sorted in x  
Default is SORT = 0
- nosymbol* - If present and not zero, no symbols will be plotted.

**Examples:**

Take values off a curve stored in waveform array 1 and plot the selected points in ascending values of X.

```

plo, 1 ; Put the data into the current window
XCUR, x, y, /sort ; Cursor the data
plot, x, y ; Plot the results

```

Similar to the first example but plot the Y- values of the actual data not the y-value from the cursor.

```

plo, 1 ; Put the data into the current window
XCUR, x, y, /sort ; Cursor the data
nel = n_elements(x) ; Determine if points selected
if nel gt 0 then begin
for i = 0, nel-1 do y(i) = get_value(1, x(i)) ; select the actual y-values
plot, x, y ; plot the values
endif

```

**XCURW**

Select values at a finite number of points in a window from a waveform. Procedure will continuously display the (x,y) location of the cursor and store the values when the mouse button is depressed. Unless SNAP = 0, the nearest point on the curve will be stored rather than the actual position of the cursor. The cursor can be set at a specific location by editing the displayed values of X and Y and hitting a carriage return in the X or Y display window.

**Note:** MX, MN, FWHM and GET\_VALUE often provide the desired information more accurately and quicker.

**format:** XCURW [, X , Y] [, dx, dy] [, /SORT] [, /NOSYMBOL] [, SNAP = snap]

where:

- x, y* - abscissa and ordinate values when the mouse button is depressed. By default, a symbol will be placed on the plot at these values.
- dx, dy* - change in the abscissa and ordinate values  
**Note:** If data is sorted then dx and dy will be recalculated.
- sort* - If present and not zero, then data will be sorted in x  
Default is SORT = 0
- nosymbol* - If present and not zero, no symbols will be plotted.
- snap* - If present and zero, then X and Y will be actual

cursor positions.

SNAP=1; Snap to X value of waveform at current Y.  
If outside the range, go to minimum or maximum Y.

SNAP=2; Snap to Y value of waveform at current X.  
If outside the range, go to maximum or minimum X.

SNAP=3; Snap to waveform using linear interpolation between points

SNAP=4; Snap to nearest actual point in the waveform.

*wdfarray*

- Store X and Y in a WDFarray given by WDFARRAY. If a valid array is not specified, then the data will be stored in an empty array if one is available or in MAXwdfarray if no arrays are empty.

Examples:

Take values off a curve stored in waveform array 1 and plot the selected points in ascending values of X.

**XCURW, 1, x, y, /sort ; Cursor the data**  
**plot, x, y ; Plot the results**

Same as first example but store XY in WDF array 2

**XCURW, 1, /sort, w=2**  
**plo, 2**

## XFR

Generates a duplicate WDF array in a second WDF array. WDF array labels, file name, etc. are also copied to the new WDF array.

**format: XFR, WD1, WD2**

where:

*wd1, wd2* - WDF array numbers:  $wd2 = wd1$

Example:

Copy WDF array 4 to array 5

**XFR, 4, 5**

## XFRSTR

Generates a duplicate structure array in a second structure array. WDF array labels, file name, etc. are also copied to the new WDF array.

**format: XFRSTR, str1, str2**

where:

*str1, str2* - Structure array numbers

Example:

Copy structure array 4 to array 5

```
XFRSTR , 4, 5
```

## XLINEOUT

Calculate a lineout of a two dimensional array at a given y value and over a specified width.

format: `line = XLINEOUT(ARRAY, YARRAY, YVALUE [, WIDTH])`

where;

LINE	- array of ordinate values for the lineout
ARRAY	- two dimensional array
YARRAY	- Ordinate values for array
YVALUE	- Ordinate value at the center of the lineout
WIDTH	Width over which the values of array are averaged to obtain the values in LINE. Default is 0.0

Example:

Obtain a horizontal lineout from ARRAY with ordinate values, Y, at a value of  $y = \text{AMAX}$ .

```
line = XLINEOUT(array, y, amax)
```

**Note:** Since width is zero, value is simple interpolation.

Obtain a horizontal lineout from PICTURE with ordinate values, Y, at a value of  $y = \text{yy}(i)$ , averaged over a width of 0.06.

```
line = XLINEOUT(picture, y, yy(i), 0.06)
```

## XPLO

Plot one WDF array versus a second WDF array at the same points in time. The WDF y-axis labels of the two arrays are used for the axis labels.

format: `XPLO, WDX, WDY [, XRANGE =xr] [, YRANGE =yr] [, TITLE =title] [, /NOGRID] [, DELTA = delta] [, SYMBOL =symbol] [, LINSTYLE =linestyle] [, COLOR = color] [, _extra = extrastuff]`

where:

<i>wdx, wdy</i>	- WDF array numbers to be plotted along the horizontal and vertical directions
<i>color</i>	- indicates the color to be used for the curve. If only the keyword (/C) is present, the curve will be red. The colors are: 0 = black, 1 = red, 2 = green, 3 = blue, 4 = magenta, 5 = orange, 6 = cyan, 7 = yellow, and 8 = white.

- linestyle* - integer indicating the type of line to be used. Values of LINE are: 0 is solid, 1 is dot, 2 is dash, 3 is dot-dash, 4 is dot-dot-dot-dash, and 5 is long dash. Default value is 0.
- nogrid* - indicates the curve should be plotted without drawing a new grid.
- delta* - Indicates a symbol is to be plotted on the curve at approximate increments given by delta.
- symbol* - indicates the type of symbol to be used. If delta is zero, about 40 symbols will be plotted and lines will connect the points. Values of *symbol* are: 1 is plus sign, 2 is asterisk, 3 is period, 4 is diamond, 5 is triangle, 6 is square, and 7 is "X". Default is 1.
- title* - string to be used as the title of the plot.
- xr, yr* - two element arrays with the first element equal to the minimum and the second element, the maximum value of the x-axis and y-axis respectively. These values will be rounded to "nice" values unless *xstyle* and/or *ystyle* have been set.
- \_extra* - any valid keywords for the *PLOT* command.

Example:

Plot WDF array 4 versus array 5 and put a symbol a 5 ns increments

```
XPLO , 4, 5, delta = 5e-9
```

## XRESET

Function: enable the cursor keys of an xterminal

format: **XRESET**

Keywords:

- VERBOSE* - If set will indicate: '% XRESET: Resetting xterm cursor keys'

## YLINEOUT

Calculate a lineout of a two dimensional array at a given x value and over a specified width.

format: **line = YLINEOUT(array, xarray, xvalue [, width])**

where:

- line* - array of ordinate values for the lineout
- array* - two dimensional array
- xarray* - Abscissa values for array
- xvalue* - Abscissa value at the center of the lineout

*width* - Width over which the values of array are averaged to obtain the values in LINE. Default is 0.0 When the width is zero, linear interpolation is used.

Example:

Obtain a horizontal lineout from *ARRAY* with abscissa values in the array, *X*, at a value of *x = AMAX*

```
line = YLINEOUT (array, x, amax)
```

Obtain a horizontal lineout from *PICTURE* with abscissa values, *X*, at a value of *x= xx(i)*, averaged over a width of 0.06

```
line = YLINEOUT(picture, x, xx(i), 0.06)
```

## ZOOMW

Provide a easy to zoom display of waveform data.

Procedure: The range and location for the zoomed image may be set in three different ways.

1. Sliders allow the maximum and minimum of the Zoomed image to be set.
2. Sliders allow the center point and the magnification to be set.
3. The center point can be set by clicking on the original image; the maginifcation will be taken from the slider value.

**Note:** Slider values may be changed by moving the sliders or by typing into the slider display windows and ending input with a "RETURN"

**Note:** Display will not be changed unless zoom endpoint changes by more than 3%

**Note:** All inputs are the same as the PLO command. See PLO for more details.

```
format: ZOOMW, wda [,ncurve] [, MAXZOOM = maxzoom] [XSIZE = xsize]
[, YSIZE = ysize] [, ZOOMRANGE = zoomrange]
[, /OVER] [, XRANGE = xr] [, YRANGE = yr]
[, GRID = grid] [, LEGEND = leg] [, NSYMBOL = nsymbol] [, /NOGRID]
[, COLOR = color] [,PSYMBOL = symbol] [, LINSTYLE = line]
[, /NLINestyle] [, DATA = data] [, TITLE = title] [, LGSP= legspace]
[,XTITLE = xtitle] [, YTITLE = ytitle ] [, NCHAR = nchar]
[, NEXTPOS = nextpos] [, THICK = thick] [, CHARSize = charsize]
[, LCHARSize = lcharSize] [, ALLSYMBOL = allsymbol] [, UNSET = unset]
[, SETDEFAULT = setdefault] [SHOWDEFAULT = showdefault ]
[, HEADER = header] [, LEFT = left] [, RIGHT = right] [_EXTRA = extrastuff]
```

where:

*wda* - WDF array number or an array of WDF array numbers

*ncurve* - number of curves to be plotted. If WDA is a single WDF array number, then WDA and the following *ncurve-1* arrays will be plotted.

*maxzoom* - maximum ZOOM parameter, Default = 100.  
Adjusting endpoints on the graph by moving the

slider or typing into the window and ending with a CR will allow the zoom value to increase another factor of about 3; however the display will only read MAXZOOM

- zoomrange* - final value of XRANGE; can be used with PLO to plot other data with the same abscissa values
- X/Ysize* - Size of windows. Default is:  
Xsize = 620, Ysize = 540 for HP  
Xsize = 550, Ysize = 480 for SUN

**Note:** Most inputs are the same as the PLO command. See PLO for more details. Exceptions include the following which are set specifically for part or all of the ZoomW command.

- legend* - legend position for the default plot. This is set to -1 for the zoomed image
- header* - used for the plot header; set to 0 for the zoom image
- grid* - Set to 1
- over* - If Ncurve gt 1 or WDA is an array, then over = 1.
- NoGrid* - Set to zero
- xstyle* - Set to 1 by default for all zoom images. Original plots will use the specified values of Xstyle or 1 if no value is provided

**Note:** SetDefault, Unset, and ShowDefault are not modified, but are generally not recommended.

**Note:** XRANGE should not be specified

Example:

Examine the details of the data in wdf array 3

**ZOOMW, 3**

Compare arrays 2, 3, 4, and 5 in detail; put the legend at relative positions of [0.5, 0.5]

**ZOOMW, 2, 4, /ov, leg = [.5, .5], /col**



## A.0 The PFIDL Structures

### A.1 PFIDL Structures

Field, particle, grid, conductor and WDF array data is stored in special data structures. The structure types are listed in Table 3.

**Table 3:**

Structure type number	Description
Type 1	Electromagnetic Fields, Charge density, Images, ...
Type 2	Particle Data (position and attribute data)
Type 3	Rectilinear Grid Data
Type 4	Conductor Data (Quicksilver)
Type 5	Integer Float Data
Type 6	Waveform arrays
Type 7	ACIS geometry slices
Type 8	Finite Element 2-D Data at fixed time
Type 9	Finite Element 2-D Data for multiple times
Type 10	Finite Element Grid Data (Alegra)
Type 101	X-ray Diagnostic Array Header

The elements of each structure is shown in Table 4.

**Table 4: Table of structure parameters**

Field	Particle	Grid	Conductor	Integer-Float	Time History	ACIS Slice	Field fe grid	Field +time fe grid	Exodus II grid	Definition
1	2	3	4	5	6	7	8	9	10	Structure type
CLAB	CLAB	CLAB	CLAB	CLAB	CLAB	CLAB	CLAB	CLAB	CLAB	Dataset comment
TLAB	TLAB	TLAB	TLAB	TLAB	TLAB	TLAB	TLAB	TLAB	TLAB	Dataset type label
FILE	FILE	FILE	FILE	FILE	FILE		FILE	FILE	FILE	File source of data
SPARE	SPARE	SPARE	SPARE	SPARE	SPARE		SPARE		SPARE	Integer array (1 or more)
NDIM	NDIM	NDIM	NDIM		NDIM = 1	NDIM	NDIM=2	2	NDIM	Number of spatial dimensions
BLOCK	BLOCK (=1)	BLOCK	BLOCK (=1)		BLOCK		BLOCK	BLOCK	BLOCK	Number of blocks of data
SIZE (NDIM, BLOCK)	NVERT	SIZE (NDIM, BLOCK)		Size(3) [n-farray, n-Flist n-farray]	NP		SIZE (NDIM, BLOCK)	SIZE	SIZE	Sizes for each dimension for each block or number of particles or time steps.
LOCX (NDIM, BLOCK +1)		LOCX (NDIM, BLOCK +1)				LOCX	LOCX	LOCX	LOCX	Array of locations for the first element of each block for each dimension; locaton of each Exocus cell
X1, X2, X3, X4 - Xi = Xi (LOCX(i-1, block))	X1, X2, X3, X4 - Xi = Xi (NVERT)	X1, X2, X3 - Xi = Xi (LOCX(i-1, block)			X1	X1, X2, X3 = Xi (LOCX (locp(im) :locp(i)-1))	X1, X2	X1, X2, X4	X1, X2, X4	Spatial arrays - data in NDIM valid arrays

**Table 4: Table of structure parameters**

Field	Particle	Grid	Con- ductor	Integer- Float	Time History	ACIS Slice	Field fe grid	Field +time fe grid	Exodus II grid	Definition
X1LAB, X2LAB, ... X4LAB	X1LAB, X2LAB, ... X4LAB	X1LAB, X2LAB, X3LAB	XLAB(3)		X1LAB	X1LAB, X2LAB, X3LAB	X1LAB, X2LAB	X1LAB, X2LAB, X4LAB		Labels for spatial arrays - NDIM valid arrays of length BLOCK
NVECT							NVECT	NVECT	NTIME	Number of vector compo- nents or time values
	NATT									Number of particle attributes
LOCV (BLOCK +1)							LOCV	LOCV	LOCV	Array of locations for the first element of each block for each vector or Cell (10)
V1, V2, V3, V4, V5, V6 - Vi = Vi* (LOCV (block))	V1, V2, V3, V4, V5, V6 - Vi = Vi (NVERT)						V1, V2, V3, V4, V5, V6 - Vi = Vi (NVERT)	V1, V2, V3, V4, V5, V6 - Vi = Vi (NVERT)		Vector components or parti- cle attributes - data in NVECT or NATT linear arrays
V1LAB V2LAB, ... V6LAB	V1LAB V2LAB, ... V6LAB				V1LAB		V1LAB V2LAB, ... V6LAB	V1LAB V2LAB, ... V6LAB		Labels for vector or attribute arrays - NDIM valid arrays of length BLOCK
TWO	TWO	TWO				TWO	TWO	TWO		Integer indicating missing dimension for 2D data or degenerate 3D data
SLICE						SLICE	SLICE	SLICE		Location of 2D data slice if known - Default = 0.0

**Table 4: Table of structure parameters**

Field	Particle	Grid	Con- ductor	Integer- Float	Time History	ACIS Slice	Field fe grid	Field +time fe grid	Exodus II grid	Definition
			SIZE (5)				SIZE	SIZE		Number of slant planes(0); planes normal to each dimension (1,2,3); and the total(4).
RANGE (2, NDIM, BLOCK+ 1)		RANGE (2, NDIM, BLOCK+ 1)	PLANE (2,3, size(4))		Range (2)	Range(2, 3)	RANGE	RANGE		Maximum and minimum for each dimension in each block or conductor plane
ORDER (NDIM, BLOCK)							ORDER = 0	ORDER = 0		Ordering of blocks in each dimension by minimum value
Connect (7, ncon- nect)**									Connect (0: locv (block)- 1)	Type 1:Block connection array Type 10: Points for each cell.
			N0(size(0 )), N1(size(1 )) N2(size(2 )), N3(size(3 ))							Plane indices for slants and normals to each dimension - Value = -1 if no planes exist.
			SLANT (3,2, size(0))							Array of the opposite cor- ners for each slant plane - Value = 0 if no slants
				FLIST						Float List

**Table 4: Table of structure parameters**

Field	Particle	Grid	Con- ductor	Integer- Float	Time History	ACIS Slice	Field fe grid	Field +time fe grid	Exodus II grid	Definition
				IArray						Integer Array
				FArray						Float Array
					X0					Initial Abscissa Value
					DX					Point Spacing (-1 = Non-Uniform)
					Yrange (2)					Ordinate [Min., Max.]
						POINT				X, Y, Z point in slice plane
						Normal				Direction of normal
						LOCP				Pointer to each conductor

\* Note: V1, V2, ...V6 are one dimensional arrays of 3D data. GET\_ARRAY will provide the appropriate 3D arrays.

\*\* The first index of the connection array is defined according to the following:

0. Connection direction 1, 2, or 3 for the normal direction
  1. Block with the smallest minimum in the normal direction
  2. Block with the minimum equal to the connection plane
  3. Low value of the first remaining dimension of the overlap
  4. High value of the first remaining dimension of the overlap
  5. Low value of the last remaining dimension of the overlap
  6. High value of the last remaining dimension of the overlap
- nconnect is the number of connections. For single block data, connect = 0.



## B.0 Command Syntax

### B.1 Comparison of WDF and structure commands

Table 5:

WDF	Structure	Description
ADD	ADDSTR	Add two arrays or an array and a constant
CHA	CHASTR	Change an array label or scale the values
DELOWDF	DELSTR	Delete an array
DIV	DIVSTR	Divide an array by another array or a constant
GETX GETY GETYF	GET_ARRAY	Obtain array or structure elements into IDL variables
I2W W2I	I2S S2I	Convert IDL arrays or a structure to a WDF array or a structure array and the inverse.
INT	INTSTR	Integrate an array or structure
PLO	PLOTSTR	Plot an array(s)
PUTX PUTY	PUT_ARRAY	Replace array or structure elements
RE	READSTR	Read a PFF dataset into an array
SUB	SUBSTR	Subtract two arrays or an array and a constant
WHELP	SHELP	Obtain information about an array(s)
WRI		Write a WDF array
SAVEPFIDL RESTOREPFIDL	SAVEPFIDL RESTOREPFIDL	Save and restore WDF arrays and structure arrays



## C.0 Quicksilver Application Datatypes

### C.1 Quicksilver Application Datatypes

The following application dataset types are in current use in Quicksilver and TwoQuick PFF output files. These are listed in comdeck *qpfparam*.

Application Dataset Type	Usage
-3	- Default type value
1	- Bounding box
2	- Conductor data
3	- Time-history data
4	- Dielectric data
5	- System grid data
11	- Electric field snapshot data
12	- Magnetic field snapshot data
13	- Current density snapshot data
14	- Charge density snapshot data
20	- Particle (only) snapshot data
21	- Particle (w/ charge) snapshot data
23	- Particle (w/ momentum) snapshot data
24	- Particle (w/ charge and momentum) snapshot data
25	- Particle (w/ charge, momentum, and integer attribute)
26	- Particle Trajectories (w/ charge, momentum, and particle number or momentum, and particle number) (Particle number is always in attribute 6)
27	- Particle (w/ charge, momentum, and kill time)
28	- Particle (w/ charge, momentum, kill time, and integer attribute.)
29	- 1D Dynaid Particle snapshot data



## D.0 TQ Build

### D.1 Overview

A graphical user interface for generation of a TWOQUICK (TQ) input deck has been written to assist the user in generating an input deck for the 2D electromagnetic particle in cell TQ code; some extensions have been added to permit limited QUICKSILVER input deck generation. This interface uses DXF (Drawing Interchange File Format) files, ASCII tabular files, or user generated ordered pairs, using either an analytic formulation or a graphical user interface, to generate a conductor surface in two dimensions. The grid is generated using a quadratic generating function which guarantees a continuous first derivative. The conductors are automatically fit to the grid, but the fit may be modified using a graphical user interface. The grid may be modified automatically to fit a particular straight line segment exactly. Marker locations may be designated for code diagnostics. The resulting geometry is output to a file for input into the TQ code. The user may specify the units to be used for the conductors and the grid; inches, mils, and centimeters are standard input options.

A typical TQ run would involve the following steps for generating an input deck for the:

1. Generate ordered pairs for each conductor.
2. Generate a uniform grid to use in editing the conductors (optional).
3. Edit the conductors by adding points, deleting points, combining, splitting or deleting conductor segments.
4. Generate a grid for the actual calculation. (Can be previous grid from #2.)
5. Fit the conductors to the grid.
6. Edit the fit of the conductors to the grid. The grid in one direction may be modified to exactly fit a sloping surface.
7. Designate marker positions for code diagnostics as a last step in the process.
8. Write the file to be used as a TQ input deck.
9. Edit the input deck to add additional diagnostics.

The data is stored in a TQ common block in IDL to minimize the number of variables which must be passed between procedures. A typical session might include the following commands where we assume conductor data is stored in a DXF file and that a uniform grid is desired:

```
file = pickfile(/read)
readdxf, x, y, loc, file
tqcond, x, y, loc
tqgrid, [0, 25], [.1, .1], /x , /inches
tqgrid, [0, 30], [.1, .1], /y
tqedit, /cond
tqfit
tqedit
```

If the grid is to be regenerated, then TQGRID and TQFIT commands can be repeated to generate a different grid and fit. Some options for conductor generation will be discussed in the next section. The gridding feature of this package is the more complex aspect of the procedure; this will be discussed in detail followed by a command dictionary for all the commands.

## D.2 Conductor Definition

The TQ conductors must be defined as a series of ordered x-y pairs. Any process which produces these ordered pairs is possible, but four techniques will be described in this section: graphical interface, ASCII data file, DXF data file, and IDL functions. After the x-y pairs are determined, they are stored in a common block using the command, TQCOND. The particular units must be the same as those chosen for the grid, but any units are acceptable.

### D.2.1 Graphical User Interface

This technique involves using a graphical user interface to add/delete points to define the conductor or conductors. Before using the graphical interface, one conductor must be specified. If the problem spans the space (0, 2) in the x-direction and (0, 4) in the y-direction, then a command of the form:

```
TQCOND, [0,2], [0,4]
```

will generate an initial conductor. A useful, but optional, additional step is to generate a grid to superimpose on the problem to assist in generating the conductor. This can be accomplished with the following command where we desire to cover the above range and domain with a grid having 20 lines per unit.

```
TQGRID, [0, 2], [1, 1]/20.0, /x
```

```
TQGRID, [0, 4], [1, 1]/20.0, /y
```

If the units are not meters and this grid geometry is desired for the final problem, then a keyword parameter must be added to scale the output parameters. The grid command will be discussed in more detail in the next section and D.4. The graphical editor is invoked with the command:

```
TQEDIT, /Conductor
```

where the conductor keyword indicates editing the conductor data. From this procedure the conductor can be completely modified by adding points, deleting points, splitting conductor segments, or joining conductor segments. At reasonable intervals TQEDIT should be exited and the command TQSAVE should be used to save the conductor parameters from unforeseen disasters.

### D.2.2 ASCII Data File

Data which has been defined as ordered pairs of x-y pairs in an ASCII data file can be input with the commands:

```
READASC, 'anode.dat', 2, x, y
```

```
TQCOND, x, y
```

where we assume that the file, 'anode.dat' contains two columns of x-y pairs. Other

formats are possible (See the READASC command in chapter 4). These two commands can be repeated for each of the conductors in the problem. TQCOND will append additional conductors to the previously defined conductors.

### D.2.3 DXF File

Various drafting packages allow drawing information to be output in the form of a DXF (Drawing Interchange File Format). A crude reader which supports the most common entities has been written to read this type of file and to process the data. To do this from IDL and store the data in the TQ common block, issue the commands:

```
READDXF, x, y, loc, 'parts.dxf'
TQCOND, x, y, loc
TQEDIT, /conductor
```

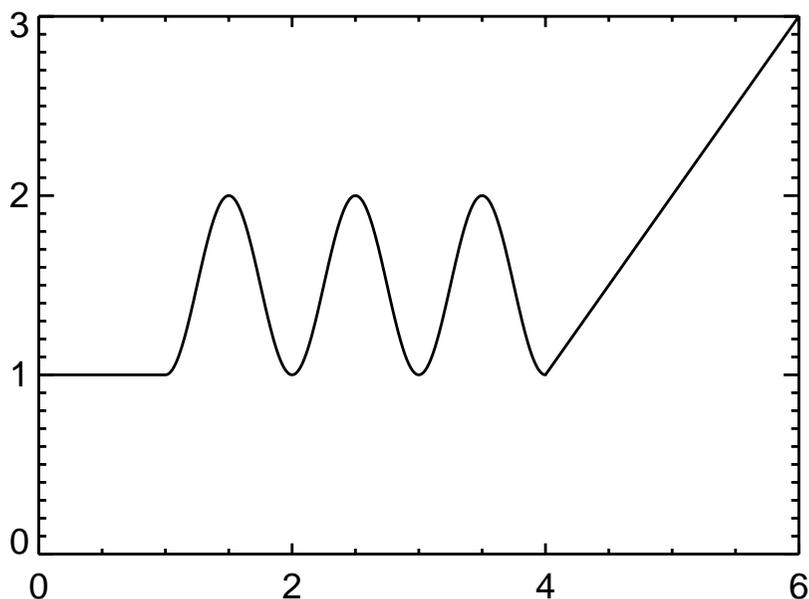
where 'parts.dxf' is the name of the DXF file. The last command allows the user to clean up the conductor configuration using a graphical user interface.

### D.2.4 IDL Functions

Simple analytic conductor configurations can be generated directly in IDL. For example, assume a cylinder of radius 1 and length 1 followed by a sinusoidally varying radius of length 3, period 1, and amplitude 0.5 with an offset of 1.5, followed by a 45 degree cone. This is sketched below and is generated with the following commands:

```
t = findgen(301)/100
x = [0, t+1, 6]
y = [1, 1.5 - 0.5*cos(2.0*!pi*t), 3]
TQCOND, x, y
```

where the phase has been adjusted to merge at (1,1). The generated geometry is below:



### D.3 Grid Generation

The grid generation uses a quadratic or a geometric grid generation function and is implemented in such a way to insure a continuous first derivative. The grid generation is implemented in TQGRID and QSGRID; section D.3.4 contains specific details. The grid generation includes a facility for scaling all spatial data if meters is not the dimension of choice. This choice of units need only be done once, but it must be done before the first grid is generated. The discussions will focus on the X axis but the same can be said for the Y axis. TQGRID supports only a quadratic generating function; QSGRID allows both quadratic and geometric generating functions. In addition QSGRID allows limited three dimensional and multiple block capabilities.

#### D.3.1 Mathematical Overview

The grid generation in TQBuild assumes that each problem can be broken into discrete regions. Each region has an initial point,  $x(0)$ , a final point,  $x(n)$ , an initial derivative,  $dx(0)$  and a final derivative,  $dx(n)$ . The gridding procedure must be continuous; therefore we require that  $x(n)$  of a preceding region equal  $x(0)$  of the current region. In addition to minimize the errors in the TWOQUICK/QUICKSILVER calculations, we require the derivatives to be continuous between the successive regions. The actual grid locations in a given region are determined from a grid generation function. Both TQGRID and QSGRID allow a quadratic generating function of the form:  $x(i) = x(0) + A*i + B*i^2$ , where A and B are constants. QSGRID also offers a geometric, or exponential grid, of the form:  $x(i) = (dx(0) / \log(R)) * (R^i - 1) + x(0)$  where  $dx(0)$  and R are constants. Both generating functions have two free parameters so but basically three characteristic quantities: the initial and final derivative and the total extent ( $x(0) - x(n)$ ) of the region. The requirement of a continuous derivative fixes one of the three parameters so the user is free to select either the other derivative or the extent of the subgrid region as the other parameter. The default procedure is to fix the total extent, that is, the end points of a region, and make a best effort to satisfy the users wishes for derivative at the free end of the interval.

The primary difference between TQGRID and QSGRID is the way that the derivatives linking adjacent regions are defined. TQGRID assumes a practical definition of the derivative,  $dx(0) = x(1) - x(0)$ , and QSGRID uses the derivative of the grid generation function at the endpoints. The TQGRID technique actually reduces the number of intervals in the previous grid region by 1 and uses the last interval as the first region in the next grid region.

Both routines use the same technique for specifying the desired grids. A vector of X-values are specified which by default define grid locations and grid region boundaries. In addition a vector containing the size of the grid region (that is, the first derivative) at each of these X-values is specified. The grid is then generated from these arrays of X and DX values. If a value of DX is negative, then the grid derivative will be set exactly to the first negative DX(i) at the corresponding location, X(i) and the grid will be built out in each direction. If more than one DX value is negative, then the value of X(j) will be varied slightly to satisfy the condition that grid derivative be exactly  $abs(DX(j))$  as near X(j) as possible.

In the next sections examples of different types of grids are presented with the

detailed equations in section 3.3.

### D.3.2 Examples

#### D.3.2.1 Uniform Grid

To make a uniform grid with values of 0.05 between 0 and 1 use the command:

```
TQGRID, /x, [0, 1], [0.05, 0.05]
```

or

```
QSGRID, /x, [0, 1], [0.05, 0.05]
```

where /x indicates the X axis.

For a uniform grid with 50 values of grid regions between x0 and x1 use the command:

```
TQGRID, /x, [x0, x1], [1.0, 1.0] * (x1-x0)/50.
```

```
QSGRID, /x, [x0, x1], [1.0, 1.0] * (x1-x0)/50.
```

```
QSGRID, /x, [x0, x1], [1.0, 1.0], g = -50
```

#### D.3.2.2 Nonuniform Grid

To produce a grid between [0, 5] which starts has an initial spacing of 0.05 at 0, has a spacing of 0.1 at near 1, a spacing of 0.25 near 2.5 and a spacing of about 0.3 at 5, the following command would be used.

```
tqgrid, /x, [0, 1, 2.5, 5], [-0.05, -0.1, -0.25, 0.3]
```

In this command the first negative value of  $dx$  sets both the value of  $dx$  and  $x$  exactly. The second and third negative  $dx$  values set the value of  $dx$  exactly allowing the value of  $x$  to vary slightly to have an integer number of grids, and the last positive value of  $dx$  allows  $dx$  to vary slightly to set the last grid line exactly at the specified value of  $x$  (5 in this example.) To have grid lines at exactly [0, 1, 2.5, 5] and an initial delta  $x$  of exactly 0.05 the command would be modified as shown below:

```
tqgrid, /x, [0, 1, 2.5, 5], [0.05, 0.1, 0.25, 0.3]
```

For these nonuniform grids the options for QSGRID are identical. To obtain a geometric variation between X locations, the above example becomes

```
qsgrid, /x, [0, 1, 2.5, 5], [0.05, 0.1, 0.25, 0.3], /g
```

#### D.3.2.3 Periodic Boundary

A periodic boundary presents a unique problem. In general a uniform grid is desired with an integral number of grids for each half cycle to insure a symmetric waveform. For the example waveform in section D.2.4, the following might be used to grid the problem:

```
tqgrid, /x, [0, 1, 4, 6], [0.1, -0.05, -0.05, 0.2]
```

```
tqgrid, /y, [0, 1, 2, 3], [0.1, -0.05, -0.05, .1]
```

where the grid is spread in the  $x$  and  $y$  directions away from the periodic portion. The second negative  $dx$  value is not necessary since the specified spacing is an integer number of intervals. The half period of the example was 0.5; thus there are about 10 points per half cycle. The option to fit the X axis to the curve can be used to make the slant almost a diagonal surface. Care should be exercised in selecting the bottom point near  $x = 4$  to

avoid changing the X axis in the periodic portion of the conductor. Pick a value of about  $x=4.4$  and the X axis fit will be reasonably good (The upper point will be at (6, 3).)

### D.3.3 Grid Details

TQGRID and QSGRID have three distinct types of regions which must be considered that depend of the constraints which are present:

1. Initial point, final point and final dx specified.
2. Initial point, final point and the initial dx specified.
3. Initial point, final dx and the initial dx specified.

Each region will now be considered and the relevant equations will be presented. In the equations below the following are assumed:

$X_0, X_n$	- initial and final or $n^{\text{th}}$ X value $X_n = X_n$ $X_0 = X_0$
$DX_0, DX_n$	- initial and final or derivatives For TQGRID grid these are defined as the practical derivatives: $DX_0 = X_1 - X_0$ For QSGRID grid these are the actual mathematical derivatives of the grid generation function.
n	- number of intervals
dist	- extent of region $\text{dist} = X_n - X_0$
findgen	- IDL floating point index generator $\text{findgen}(n) = [0.0, 1.0, 2.0, 3.0, \dots, n-1]$
Aup	- A value from larger X values
Rmin, Rmax	- minimum and maximum allowable values for R for a geometric grid

#### D.3.3.1 Grid Type 1: $X_0, X_n, DX_n$

This type of region occurs for the first region with a negative value of DX (unless it is the first region.) From out grid matching conditions for both QSGRID and TQGRID,

$$DX_n = DX_0'$$

where the primes refer to the previous grid region. For the first grid region  $DX_n$  is the absolute value of the negative DX value

##### D.3.3.1.1 QSGRID: Quadratic Grid

First calculate the number of intervals using the suggested value of  $DX_0$ :

$$n = \text{round}(2.0 * \text{dist} / (DX_0 + DX_n))$$

Calculate A and B:

$$B = (DX_n - \text{dist}/n)/n$$

$$A = 2 * \text{dist}/n - DX_n$$

If  $-A/2/B$  is greater than zero, insure that  $n$  is less than this value.

Calculate X's:

$$X = ((B * \text{findgen}(n+1) + A) * \text{findgen}(n+1) + X_o)$$

#### D.3.3.1.2 QSGRID: Geometric Grid

First estimate the ratio of adjacent cells,  $R$ ,

$$R = \exp((DX_n - DX_o) / \text{dist}).$$

Limit  $R$  to an acceptable range, and, if it must be changed, change  $DX_o$ .

$$R = R_{\min} < R < R_{\max}$$

$$DX_o = DX_n - \text{dist} * \log(R)$$

Calculate  $n$

$$n = \text{round}(\log(DX_n / DX_o) / \log(R))$$

Modify  $R$  to account for rounded value of  $n$ .

Calculate X's:

$$X = DX_o / \log(R) * (R^{\text{findgen}(n+1)} - 1.0) + X_o$$

#### D.3.3.1.3 TQGRID

First calculate the number of intervals using the suggested value of  $DX_o$ :

$$n = \text{round}(2.0 * \text{dist} / (DX_o + DX_n))$$

Calculate  $A$  and  $B$ :

$$DX_o = 2 * \text{dist} / n - DX_n$$

$$B = (DX_n - DX_o) / (n-1) / 2 = (DX_n - \text{dist}/n) / (n-1)$$

$$A = DX_o - B$$

Calculate X's:

$$X = ((B * \text{findgen}(n+1) + A) * \text{findgen}(n+1) + X_o)$$

#### D.3.3.2 $X_o$ , $X_n$ , and region matching conditions for the lower points

This type of grid occurs for the default situation where the grid lines fall exactly on the specified  $X$  values and the grid must match the preceding subgrid region. If this is the first grid region, then  $DX_o$  is the first value of  $DX$ .

##### D.3.3.2.1 QSGRID: Quadratic Grid

From out grid matching conditions,

$$DX_o = A' + 2 * B' * n' = A$$

where the primes refer to the previous grid region.

First calculate the number of intervals using the suggested value of  $DX_n$ :

$$n = \text{round}(2.0 * \text{dist} / (DX_o + DX_n))$$

Calculate  $B$ :

$$B = (\text{dist} / n - A) / n$$

If  $-A/2/B$  is greater than zero, insure that  $n$  is less than this value.

Calculate X's:

$$X = ((B * \text{findgen}(n+1) + A) * \text{findgen}(n+1) + X_o)$$

## D.3.3.2.2 QSGRID: Geometric Grid

From out grid matching conditions,

$$DX_o = DX_o' * (R')^n$$

where the primes refer to the previous grid region.

Next estimate the ratio of adjacent cells, R,

$$R = \exp((DX_n - DX_o) / \text{dist}).$$

Limit R to an acceptable range, and, if it must be changed, change the approximate value of DX<sub>n</sub>.

$$R = R_{\min} < R < R_{\max}$$

$$DX_n = DX_o + \text{dist} * \log(R)$$

Calculate n

$$n = \text{round}(\log(DX_n / DX_o) / \log(R))$$

Modify R to account for rounded value of n.

Calculate X's:

$$X = DX_o / \log(R) * (R^{\text{findgen}(n+1)} - 1.0) + X_o$$

## D.3.3.2.3 TQGRID

From out grid matching conditions,

$$DX_o = DX_n'$$

where the primes refer to the previous grid region.

Calculate the number of intervals using the suggested value of DX<sub>n</sub>:

$$n = \text{round}(2.0 * \text{dist} / (DX_o + DX_n))$$

Calculate A and B:

$$DX_n = 2 * \text{dist} / n - DX_o$$

$$B = (DX_n - DX_o) / (n-1) / 2 = (\text{dist} / n - DX_o) / (n-1)$$

$$A = DX_o - B$$

Calculate X's:

$$X = ((B * \text{findgen}(n+1) + A) * \text{findgen}(n+1) + X_o$$

D.3.3.3 X<sub>o</sub>, DX<sub>n</sub> and region matching conditions for the lower points.

This type of grid occurs when two or more negative DX values are specified for the grid sub regions.

## D.3.3.3.1 QSGRID: Quadratic Grid

From out grid matching conditions,

$$DX_o = A' + 2 * B' * n' = A$$

where the primes refer to the previous grid region.

First calculate the number of intervals using the required value of DX<sub>n</sub>:

$$n = \text{round}(2.0 * \text{dist} / (DX_o + DX_n))$$

Calculate B:

$$B = (DX_n - DX_o) / (n + n)$$

If  $-A/2/B$  is greater than zero, insure that  $n$  is less than this value.

Calculate X's:

$$X = ((B * \text{findgen}(n+1) + A) * \text{findgen}(n+1) + X_o)$$

#### D.3.3.3.2 QSGRID: Geometric Grid

From out grid matching conditions,

$$DX_o = DX_o' * (R')^n$$

where the primes refer to the previous grid region.

Next estimate the ratio of adjacent cells,  $R$ ,

$$R = \exp((DX_n - DX_o) / \text{dist}).$$

Limit  $R$  to an acceptable range, and, if it must be changed, change the approximate value of  $DX_n$  and print a warning to the user.

$$R = R_{\min} < R < R_{\max}$$

$$DX_n = DX_o + \text{dist} * \log(R)$$

Calculate  $n$

$$n = \text{round}(\log(DX_n / DX_o) / \log(R))$$

Modify  $R$  to account for rounded value of  $n$ .

$$R = (DX_n / DX_o)^{1/n}$$

Calculate X's:

$$X = DX_o / \log(R) * (R^{\text{findgen}(n+1)} - 1.0) + X_o$$

#### D.3.3.3.3 TQGRID

From out grid matching conditions,

$$DX_o = DX_n'$$

where the primes refer to the previous grid region.

Calculate the number of intervals using the suggested value of  $DX_n$ :

$$n = \text{round}(2.0 * \text{dist} / (DX_o + DX_n))$$

Calculate  $A$  and  $B$ :

$$B = (DX_n - DX_o) / (n-1) / 2$$

$$A = DX_o - B$$

Calculate X's:

$$X = ((B * \text{findgen}(n+1) + A) * \text{findgen}(n+1) + X_o)$$

## D.4 TQBuild Command Reference Dictionary

### READDXF

Read a DXF file. The DXF file will be read and the known entities stored as conductor segments. The procedure will try to connect as many segments as possible using a default or specified resolution.

**Caution:** This is a preliminary version. Not all entity types are supported.

**format:** `readdxf, x, y, loc, file, nosort = nosort, resolution = resolution, layers = layers, plotall = plotall, insertblocks = insertblocks`

where:

- x* - abscissa values of the conductors
- y* - ordinate values of the conductors
- loc* - pointer array to the first element of each conductor  
For conductor n:  $x = x(\text{loc}(i-1):\text{loc}(i)-1)$
- file* - file name of the DXF file
- nosort* - if present and not zero, do not sort data
- resolution* - optional parameter to specify how close points must be to connect them into a single conductor.  
Default is:  $\max([x, y]) / 4.0e3$
- layers* - returns the layer value for each entity. TQFILTER can be used to select certain layers.
- plotall* - if present and not zero, plots the insert blocks with a value of `!p.multi = [0,2,2]` unless `!p.multi` has a nonzero value on entry
- insertblocks* - returns a structure of the insert blocks. Values are:  
x - xvalues  
y - yvalues  
loc - pointer to each conductor segment  
layer - layer value  
point - pointer to loc for each insert block  
name - insert block names

**Note:** Too small a value of resolution will result in disconnected conductor segments; too large a value will result in misconnected values.

Examples:

Read data from a file, 'raised\_cone.dxf'

`READDXF, x, y, loc, 'raised_cone.dxf'`

Read data from a file in '/users/scratch/'

`file = pickfile(/read, path = '/users/scratch')`

`readdxf, x, y, loc, file`

## QSEEDIT

This is designed to edit a conductor data for QS input, generate a grid, or edit the grid fitted points for tq input.

**format:** `format: qsedit, conductor = conductor, grid = grid, reset = reset, fast = fast, xsize= xsize, ysize= ysize, tquick = tquick`

where:

- conductor - If set, indicates editing conductor data
- grid - If set, generate a grid interactively.
- fast - If set, use triangles rather than circles for points  
This will speed up redrawing complex geometries.  
default is fast = 0  
fast = 1 - unfilled diamonds  
fast = 0 - filled circles for the points
- TQUICK - If set, then tqfit will be used for the fit to the grid.  
Default: TQUICK = 0, Use QSFIT for the fit.
- reset - If set, reduce the window to the conductor area.

All data must be in the TQ common block

NOTE: XCUR will print out points to the terminal if the mouse button is pressed.

Example:

Generate conductor data from scratch for QUICKSILVER.

qscond, [0, 10], [0, 1] ; define a region of interest

qsgrid, [0, 10], [.1, .1], /x ; define a reference grid

qsgrid, [0, 1], [.1, .1], /y

qsedit, /cond ;edit the conductor data

(if desired generate a new grid)

qsedit, /grid

qsfit ; Fit the data to an existing grid ( or use fit button in qsedit, /grid)

qsedit ; Edit the grid fitted conductors and add markers

qswrite ; write the output to a file

Generate conductor data from points for TWOQUICK.

tqcond, [0, 0,1], [0.1, 1, 10] ; define a region of interest

qsedit, /grid, /tq ; generate a grid interactively

tqfit ; Fit the data to an existing grid ( or use fit button in qsedit, /grid, /tq)

tqedit ; Edit the grid fitted conductors, add markers and write to afile

## QSFIT

This procedure fits conductors stored in a TQ common block ( from a TQCOND command) to the existing grid. Similar to TQFIT but slant surfaces are not allowed.

format: QSFIT

## QSGRID

This procedure generates a grid with a continuous first derivative of the the grid generation function. (See Section D.3) Either a quadratic or a geometric generating function can be specified on a given region. Depending on the function chosen, the grid locations are given by the equation:

$$X_n = X_0 + A*n + B*n^2$$

or

$$X_n = X_0 + Dx/\log(R) * (R^n - 1).$$

For a continuous first derivative, both X and DX cannot be specified exactly everywhere. For adjacent grids the following restrictions on A and B are enforced:  $(A + 2*B*n)_{\text{lower}} = (A)_{\text{upper}}$  where lower and upper refer to adjacent grid regions with smaller and larger values of X respectively.

The procedure performs the following steps:

1. All values of  $x < x(0)$  are deleted with the corresponding dx values
2. X and DX values are sorted as ordered pairs on the value of X.
3. If necessary GRIDTYPE is changed into an array with at least as many elements as X and DX.
4. Negative GRIDTYPE values are replaced and the corresponding changes made to DX.
5. Points which do not satisfy the criteria:  $X(i+1) - X(i) > 0.8 (DX(i) + DX(i+1))$ , are eliminated.
6. The grid is generated.

If all DX are positive:

Start with smallest X and generate grid using an integer number of grids points and a best effort to match the specified DX values.

If one or more negative DX values, then:

Use the first negative DX value and the corresponding X value to generate a grid toward smaller X values. In generating this grid, the end points are exact, and a best effort to match the specified DX values is made. The grid is then generated toward larger values by using an integer number of grid points and a best effort to match the specified DX values if DX(i) is positive or the endpoint is varied slightly and the endpoint derivative is given exactly by the absolute value of DX(i).

Grid equations are described in section D.3.3.

**format:** QSGRID, X, DX, GX, GRIDTYPE = gridtype, RATIO = ratio,  
 AXES = ases, BLOCKS = blocks,  
 INCHES = inches, MILS = mils, CM= cm, MULTIPLIER = multiplier,  
 XAXIS = xaxis, YAXIS = yaxis, QUIET = quiet

where:

- |          |   |
|----------|---|
| x        | - an array of grid locations at which the grid spacing is defined   |
| dx       | - an array of grid spacing desired at corresponding grid x locations                                      |
| gx       | - an optional output array of grid locations generated by this command                                    |
| gridtype | - an integer or an array of integers, which specify the type of grid generating over each subgrid region. |

The meaning of GRIDTYPE is as follows:

0 Normal quadratic grid generation scheme.

-N If the first grid region, uniform spacing with N grid spaces. If it connects to another grid region, the upper DX value is set to the grid spacing/N

Negative GRIDTYPE values are set to the  $\max(\text{GRIDTYPE}) > 0$

+N Use a geometric grid to over the grid region. Default is 0.

- axes - If present, the grid axis for output will be i, j or k depending on whether AXES is 1, 2, or 3. Default is 'i' if xaxis is set or 'j' if yaxis is set.
- BLOCKS - If present, each grid region will be given the indicated block designation. If there are more regions than blocks, the last block will be used for all remaining regions. Default is BLOCKS = 1
- ratio - smallest value of the ration between adjacent cells with a geometric grid generation function. The largest value is 1/ratio. Default is 0.9
- inches - If present and not zero, the input units are taken as inches.
- mils - If present and not zero, the input units are taken as mils =  $1.0e-3$  \*inches
- cm - If present and not zero, the input units are taken as centimeters.
- multiplier - If present and not zero, the value will be used to scale the data to meters for input into TQ/QS. Default value = 1.0  
**Note:** Multiplier is set to 0.0254, 0.0000254, or 0.01 if inches, mils or cm are set.  
**Note:** The units must be set with the first grid generated and will continue to be used for all future grids until changed.
- xaxis - If present and not zero, the x-axis is generated.
- yaxis - If present and not zero, the y-axis is generated.
- QUIET - If keyword is set, no plots or informational output will be made.

Example:

Use of gridtype:

Assume  $X = [0, 1, 2, 3]$ ,  $DX = [1, 1, 1, 1]$ ,  $G = -[10, 20, 10]$

This is would be the same as:

$X = X$ ,  $DX = [1, -.1, .05, .1]$ ,  $G = 0$

Explanation: dx(1, 2, 3) are set to the the X-spacing divided by the respective G. Because DX(0) is positive, it is changed to dx(1) and dx(1) is set negative.

Assume X = [0, 1, 2, 3], DX=[.5,1,1,1], G= [1, -20, -10]

This is would be the same as to:

X= X, DX= [.5, .05, -.05, .1], G = [1, 1, 1]

Explanation: dx(2, 3) are set to the the X-spacing divided by the respective G. Because DX(1) is positive, it is changed to dx(2) and dx(2) is set negative.

**Note:** Because of the default limit of ratio, this particular example will have an initial derivative of 0.149679 because that is as large as it can grow.

Assume X = [0, 1, 2, 3], DX=[-0.1,1,1,1], G= [-10, -20, -10]

This is would be the same as to:

X= X, DX= [-0.1, .1, .05, .1], G = 0

Explanation: dx(1, 2, 3) are set to the the X-spacing divided by the respective G. Because DX(0) is already negative, we assume that it is to remain fixed.

Assume X = [0, 1, 2, 3], DX=[-0.1,1,1,1], G= [-10, -20, -10, 1]

This is would be the same as to:

X= X, DX= [-0.1, .1, .05, .1], G = 1

Explanation: Same as above, but G is set to 1 because the maximum of G is 1.

Assume X = [0, 1, 2, 3], DX=[0.1,1,1,1], G= [-20, -20, -10, 1]

This is would be the same as to:

X= X, DX= [0.05,-0.05, .05, .1], G = 1

Explanation: Same as the second example regarding DX but because dx(0) is positive, it is changed to dx(1) and dx(1) is set negative.

Complete examples:

Generate a uniform y grid from 0 to 10 with spacing of 0.05 and units of inches.

**qsgrid, [0, 10],[0.05, 0.05], /inches, /y**

Generate a grid which varies approximately from 0.1 to 0.01 to 0.1 in going from 0 to 5 to 10 with units of cm

**qsgrid, [0, 5, 10], [.1, .01, .1], /cm, /x**

For the above example, eliminate the grid change at x = 5 and replace it with a uniform grid between 3 and 6 of 0.02 exactly.

**tqhelp, x = x, dx = dx**

**x(1) = -1 ; Eliminate old value by setting smaller than x(0)**

**x = [x, 3, 6] & dx = [dx, -0.02, -0.02] ; Add new x and dx values**

**qsgrid, x, dx, /x**

**Note:** Since (6-3) is exactly divisible by 0.02 then a value of dx of [dx, 0.02, -0.02] will yield identical results.

Generate a grid which has a dx of exactly 0.001 at 4 and is 0.01 near 10 and is about 0.01 at 0 for the y axis with units of mils

`qsgrid, [0, 4, 10], [0.01, -0.001, -0.01], /mils, /y`

## QSWRITE

This procedure writes a file which can be included into a QS geometry input deck.

**format:** QSWRITE

## TQCOND

This procedure stores conductor data in the TQ common block. Conductors can be replaced or deleted with this command. By default, conductors are added to existing conductors in the common block.

**format:** TQCOND,x, y, locx, replace = replace, delete= delete, reset = reset

where:

- x* - abscissa values of the conductors
- y* - ordinate values of the conductors
- loc* - pointer array to the first element of each conductor  
For conductor n:  $x = x(\text{loc}(n-1):\text{loc}(n)-1)$   
**Note:** *loc* is not required if only 1 conductor is input.
- replace* - Use the x-y conductor pairs to replace one or more conductors.
- delete* - Delete one or more conductors. *x*, *y* and *loc* are ignored.
- reset* - set the common block to initial conditions. Used if a new conductor geometry must be input in the same IDL session.

**Note:** Conductors are numbered: 1, 2, 3, ...

**Note:** Conductors are stored in a conductor structure. The tags for this structure are explained in TQHELP.

Examples:

Add a conductor in the TQ common block between (0,0) and (1,3).

`TQCOND, [0,1], [0,3]`

Read data from a dxf file, 'data.dxf', convert the data to cm and store in the TQ common block.

`readdxf, a, b, l, 'data.dxf'`

`a = a*2.54 & b = b*2.54`

**tqcond, a, b, l**

## TQEDIT

This is the main procedure for editing TQ input.

In the conductor mode, points may be added or deleted from conductors, conductors may be split, joined or deleted.

In the default mode, actual grid points defining the conductors may be changed, added, or deleted. Marker positions may be set and the entire set of data can be output to a file. Use TQSAVE to save intermediate results.

**format:** `tqedit, conductor = conductor, fast = fast, reset = reset`

where:

- |           |   |
|-----------|---|
| conductor | - If present and not zero, the theoretical conductor can be edited.   |
| fast      | - If present and not zero, actual points will not be plotted. This allows the screen to be refreshed much more quickly with complex geometries. |
| reset     | - If present and not zero, rescale the grid from the previous edit.   |

Example:

Edit a theoretical conductor

`TQEDIT, /cond`

Edit fitted data, removing bad points, and defining markers for diagnostics

`tqedit`

## TQFILL

Generate an array with integer values corresponding to different conductors or dielectrics.

Restrictions: Conductors must be closed and have no illegal points.

**format:** `TQFILL, array, x, y`

where:

- |       |   |
|-------|---|
| array | - Output array for the conductors and dielectrics |
| x, y  | - Abscissa and ordinates for the array            |

Example:

Generate a filled array and plot it for the current TQ data.

`TQFILL, arr, x, y`

`contour, arr, x, y, /fil`

## TQFILTER

Fit conductors to a grid.

**format:** TQFILTER, x, y, loc [, layers] [, layers = keeplayers] [, xrange = xrange] [, yrange = yrange] [, xzero = xzero] [, yzero = yzero] [, zero = zero] [, undo=undo] [, quiet = quiet]

where

x	- X conductor values
y	- Y conductor values
loc	- pointer to the conductor arrays
layers	- optional array for the layers of each conductor element
keeplayers	- optional array of desired layers.
xrange*	- desired X-range of conductors
yrange*	- desired Y-range of conductors
* <b>Note:</b> If a single value is provided, the cursor will be used.	
* <b>Note:</b> Any conductors not entirely within xrange/yrange will be eliminated.	
xzero	- set minimum x to xzero value
yzero	- set minimum y to yzero value
zero	- set minimum x and y to zero value
undo	- revert to the last x, y, loc, levels values passed to tqfilter
quiet	- if present and not zero, no plot will be make of the final result

Example:

Restore previous values

**tqfilter, a, b, c, lev, /undo**

Take only conductors between (0, 0) and (6, 4)

**tqfilter, x, y, loc, xr = [0, 6],yr = [0, 4]**

Take only layers [0, 1]

**tqfilter, x, y, loc, lev, layers = [0, 1]**

Take the set of x, y, loc and layer values from previous call to tqfilter and filter for levels [0, 1, 6]

**tqfilter, x, y, loc, lev, lay = [0, 1, 6], /undo**

## TQFIT

Fit conductors to a grid.

**format:** TQFIT

Example:

Fit conductors to a grid.

tqfit

## TQGRID

This procedure generates a grid with a continuous first derivative using a quadratic generating function. Thus the grid locations are given by the equation:  $X_n = X_0 + A*n + B*n^2$ . For a continuous first derivative, both X and DX cannot be specified exactly everywhere. For adjacent grids the following restrictions on A and B are enforced:  $(X_n - X_{n-1})_{lower} = (X_1 - X_0)_{upper}$  where lower and upper refer to adjacent grid regions with smaller and larger values of X respectively.

The procedure performs the following steps:

1. All values of  $x < x(0)$  are deleted with the corresponding dx values
2. X and DX values are sorted as ordered pairs on the value of X.
- 3.

If all DX are positive:

Start with smallest x and generate grid using an integer number of grids points and a best effort to match the specified DX values.

If one or more negative DX values, then:

Use the first negative dx value and the corresponding x value to generate a grid toward lower values. In generating the grid toward smaller x values, the end points are exact, and a best effort to match the specified dx values is made. The grid is then generated toward larger values by using an integer number of grid points and a best effort to match the specified dx values if dx(i) is positive or the endpoint is varied slightly and the interval is given exactly by the absolute value of dx(i).

Grid details are described in section D.3.4.

**format:** tqgrid, x, dx, gx, inches = inches, mils = mils, cm = cm,  
multiplier = multip, xaxis = xaxis, yaxis = yaxis

where:

- |        |  |
|--------|--|
| x      | - an array of grid locations at which the grid spacing is defined      |
| dx     | - an array of grid spacing desired at corresponding grid x locations   |
| gx     | - an optional output array of grid locations generated by this command |
| inches | - If present and not zero, the input units are taken as inches.        |
| mils   | - If present and not zero, the input units are taken as                |

- mils = 1.0e-3 \*inches
- cm - If present and not zero, the input units are taken as centimeters.
- multiplier - If present and not zero, the value will be used to scale the data to meters for input into TQ.  
Default value = 1.0  
**Note:** Multiplier is set to 0.0254, 0.0000254, or 0.01 if inches, mils or cm are set.  
**Note:** The units must be set with the first grid generated and will continue to be used for all future grids until changed.
- xaxis - If present and not zero, the x-axis is generated.
- yaxis - If present and not zero, the y-axis is generated.

**Example:**

Generate a uniform y grid from 0 to 10 with spacing of 0.05 and units of inches.

**tqgrid, [0, 10],[0.05, 0.05], /inches, /y**

Generate a grid which varies approximately from 0.1 to 0.01 to 0.1 in going from 0 to 5 to 10 with units of cm

**tqgrid, [0, 5, 10], [.1, .01, .1], /cm, /x**

For the above example, eliminate the grid change at x = 5 and replace it with a uniform grid between 3 and 6 of 0.02 exactly.

**tqhelp, x = x, dx = dx**

**x(1) = -1 ; Eliminate old value by setting smaller than x(0)**

**x = [x, 3, 6] & dx = [dx, -0.02, -0.02] ; Add new x and dx values**

**tqgrid, x, dx, /x**

**Note:** Since (6-3) is exactly divisible by 0.02 then a value of dx of [dx, 0.02, -0.02] will yield identical results.

Generate a grid which has a dx of exactly 0.001 at 4 and is 0.01 near 10 and is about 0.01 at 0 for the y axis with units of mils

**tqgrid, [0, 4, 10], [0.01, -0.001, -0.01], /mils, /y**

**TQHELP**

Return data in TQ common block.

**format: tqhelp, x=xout, dx=dcout, y=yout, dy=dyout, cond=condout, gx=gxout, gy=gyout**

where:

- x, dx - TQ grid generation parameters for the X-grid
- y, dy - TQ grid generation parameters for the Y-grid

cond - conductor structure. Current tags are:

- ncon - Number of conductors
- lcfits - Logical fit parameter
- x, y - Theoretical conductor X, Y-values
- locx - Pointer array to each conductor in x, y
- ix, iy - X, Y grid indices for the fitted conductors
- loci - Pointer array to each conductor in ix, iy
- iplotx, iploty, locp - same as ix, iy, loci but includes all intermediate grid values on diagonal surfaces.
- badxy, locb, nbad - bad points for each conductor a pointer array for each conductor and the total number.

gx, gy - X and Y grid values.

Example:

Get the x and dx values for the current grid

`tqhelp, x= x, dx= dx`

## TQRESTORE

Save the data stored in WDF arrays and in structure arrays

**format:** TQRESTORE, file, \_EXTRA = extrastuff

where:

file - File in which the data is stored  
Default is 'tqsave.sav'

\_EXTRA - Any additional keywords recognized by RESTORE

Example:

Restore the data from a previous session

`tqrestore`

Restore the data from the session in file 'qs5581.sav'

`tqrestore, 'qs5581.sav'`

## TQSAVE

Save the data stored in TQ common blocks

**format:** TQSAVE, FILE = file, \_EXTRA = extrastuff

where:

file - File in which the data is stored  
Default is 'tqsave.sav'

**\_EXTRA** - Any additional keywords recognized by SAVE

Example:

Save the data from the current session

**tqsave**

Save the data from the current session in file 'qs5581.sav'

**tqsave, file = 'qs5581.sav'**



## E.0 Convolutions

An example of these convolutions demonstrates their application. Assume a step function of the form:  $D = 1$  for  $x = .3$  to  $0.6$  and  $0$  elsewhere on the interval  $[0, 0.999]$  and a response function of the form  $R = 100*t*exp(-50*t)$  in the interval  $[0, 0.198]$ . This is done with the following:

```
a = findgen(1000) & b = 0*a & b(30:60) = 1
i2w, a/1000,b, 1
c = findgen(100)/500
e = 100*c*exp(-50*c)
i2w, c, e, 2
conv, 1, 2, 3
lab, 1, 'Data'
lab, 2, 'Response'
lab, 3, 'Conv' & plo, [3, 2, 1], /ov, leg = [.7, .9], /ns, yr= [0, 1.05], /yst, tit= ''
!p.multi = [0, 3, 2] & get_resp, 3, 1, 4 & com, 2, 4, amp = amp & mul, 4, 0, amp
plo, [4,2], /ov,leg = -1, /ns, yr= [0, 1.05], /yst, tit= 'Response',xr =[0, .2]
decon, 3, 4, 5 & lab, 5, 'Data from Calc'
decon, 3, 2, 6 & lab, 6, 'Data from Orig'
plo, [1, 5],/ov,leg = -1, /ns, yr= [0, 1.05], /yst, tit= 'Data from Calc'
plo, [1, 6],/ov,leg = -1, /ns, yr= [0, 1.05], /yst, tit= 'Data from Orig'
sub, 4, 2, 11 & sub, 1, 5, 12 & sub, 1, 6, 13
lab, 11, 'Difference in Calculated Response' & lab, 12, 'Difference from Calc'
lab, 13, 'Difference from Orig' & for i = 11, 13 do plo, i
```

