IGORE User Utility Reference Manual

Version 7.4

Gamma Ray Observatory
Oriented Scintillation Spectrometer Experiment

Naval Research Laboratory
Washington D. C.

NRL Document #926-224
Release Date: May 22, 1998

Editor: W. R. Purcell, NU

Contents

Chapter 1

Overview

1.1 Introduction to IGORE

This manual is the user utility reference manual for IGORE (Interactive GRO/OSSE Reduction Environment), the primary GRO/OSSE data analysis package. The IGORE system is based on the commercially available software package IDL, and retains all the functionality of IDL while adding specific capabilities which are required when processing the large amounts of data necessary during GRO/OSSE data analysis.

The IDL basis of the IGORE analysis system provides IGORE with an interactive programming environment for the analysis and display of data. IDL also provides the ability to easily add user defined procedures and functions to the core IGORE system. In fact, many of the IGORE commands are procedures or functions written in IDL which have been incorporated into the core IGORE analysis system.

1.2 What's in this Manual

This manual is the user utility reference guide for the IGORE system and is designed for the experienced IGORE user. Novice users should review the *IGORE User's Guide* to become familiar with the operational concepts of IGORE. The primary purpose of this manual is to provide a description of the IGORE system procedures and functions which will be of use for typical OSSE data analysis. See the *OSSE Data Analysis User's Guide* for additional examples and more details regarding the use of these procedures and functions in the analysis of OSSE data.

This manual does not describe native IDL commands; a description of the IDL system and syntax can be found in the *IDL User's Guide* or using the IDL on-line help command by typing `?` at the `Igore>` prompt. On-line help for many of the commands described in this document can either be accessed via the command IGRHELP at the `Igore>` prompt, or using the IDL library documentation facilities `DOC_LIBRARY` or `XDL`. The latter requires X-Window support at the user's terminal.

1.3 Syntax Conventions

The following syntax conventions are used in this manual:

- The main IGORE prompt is printed as `Igore>`. The actual IGORE prompt may differ, depending on the local system configuration.
- IGORE commands, parameters and keywords are specified using San Serif font.
- IDL commands, parameters and keywords are specified using `Typewriter font.`
- Square brackets ([ ]) are used to indicate either that the enclosed items are optional parameters or, in the examples, as the IDL array concatenation operator. The intended use should be clear from the context.
- Curly brackets ({ }) are used in the examples to provide parenthetical comments and are not part of the command line.

Unless otherwise stated, all user inputs are assumed to be terminated with a carriage return.

1.4 Applicable Documents

The reader may find some of the following documents useful, or even essential for analysis of OSSE data:

- *OSSE Data Analysis User's Guide* – OSSE Document #926-225.
- *IGORE User's Guide* – OSSE Document #926-175.
- *IGORE Internal/System Utility Reference Manual* – OSSE Document #926-226.
- *Spectral Data Base Header Definition* – OSSE Document #926-227.
- *IGORE Programmer's Guide* – OSSE Document #926-176.
- *IDL User's Guide*[1]
- *IDL Reference Guide*[1]
- *IDL Internals*[1]

[1] Copyright Research Systems, Inc. IDL is a trademark of Research Systems, Inc.

1.5 Acronyms and Glossary

AU	Analyzable Unit
DTW	Digital Time Window
FDB	Fit Data Base
FDR	Fit Data Record
IDL	Interactive Data Language
IGORE	Interactive GRO/OSSE Reduction Environment
OSSE	Oriented Scintillation Spectrometer Experiment
PDF	Pulsar Data File
PSF	Pulsar Selection File
SDB	Spectral Data Base
SDR	Spectral Data Record
TSF	Telemetry Scalar File

Chapter 2

Introduction

The applications described in Chapter 3 are those which would be most useful for a typical IGORE user. Some of these applications are general purpose additions to the IDL core (*e.g.*, CREREC), while others have been specifically designed to support OSSE analysis (*e.g.*, SDB$OPEN). The applications are either part of the IGORE executables, IDL procedures, or a combination of both.

The OSSE spectral analysis applications support SDB header version 7 only, unless otherwise specified. The current definitions of the IGORE SDB and the most common SDB AUX records are given in Chapter 5.

It is important to note that, for many of the spectral analysis applications, the indirect SDR fields (*e.g.*, the HDR, DAT and ERR fields, etc.) **need not be resident in memory** to successfully execute the application, as long as valid FILE and TAG fields are available in the SDR. If the required indirect field is not available in memory (*e.g.*, it has not been read into the SDR with either SDB$READ or SDR$GET), the application will attempt to read the required information directly from the SDB file using the FILE and TAG fields contained in the SDR. The search for the SDB file using the FILE field is performed in the following sequence:

- The current directory is searched using the string FILE.
- The current directory is searched using the string FILE and with the file type ".SDB".
- The directory path logical IGR$SDB is used with the string FILE.
- The directory path logical IGR$SDB is used with the string FILE and with the file type ".SDB".
- The message "Error opening SDB file." is reported.

Note that the applications SDB$OPEN and SDR$GET search for the SDB file in the same order. The application FILE_INFO may be used to determine which files are open and the associated logical unit.

Table 2.1: Valid MODE Parameter Values

Value	Action
0	Use data in memory, if available. Otherwise read data into temporary memory buffer.
1	Use data in memory, if available. Otherwise read data into permanent memory buffer.
2	Clear all record fields and read data from file into permanent memory. (Not applicable to some routines.)
3	Read additional data from file into permanent memory, overwriting any data in memory.(Not applicable to some routines.)
ELSE	Use data in memory, if available. Otherwise read data into temporary memory buffer. (Same as MODE = 0)

Many of the SDR applications below have an optional MODE parameter which indicates how the indirect records and fields are to be stored in the SDR. Table 2.1 lists the valid MODE parameter values. For most of the applications described in Chapter 3, the default value of MODE is 0. Certain MODE parameter values are not uniformly applicable to all routines.

IGORE creates, maintains, and uses records in a nested hierarchical system, allowing record structures that are more complex than the structures allowed by IDL. These IGORE records are not directly accessible to IDL. Instead, IDL variables are created as pointers to these IGORE records. It is therefore important to note the distinction between IGORE records and structures versus IDL records and structures.

A subset of the IGORE routines can be used to move data between IGORE and IDL variables. The routines include; GETFLD, PUTFLD, GETAUX, PUTAUX, GETAUXREC and PUTAUXREC.

Many of the routines that create and manipulate IGORE records accept optional keywords, including NAME and /UNIQUE. These keywords allow the user to explicitly specify the name of an IGORE record, or to force a unique name to be generated, respectively. By default IGORE record names are assumed to match the names of the IDL variables used as record pointers.

See the *IGORE User's Guide* for a full discussion of the IGORE philosophy and the related usage considerations.

Chapter 3

User Reference

ANGSEP

This function returns the separation angle between two vectors.

Calling Sequence

separation = ANGSEP(lon1, lat1, lon2, lat2)

[This function is written in IDL]

Arguments

LON1
The longitude-like coordinate of the first vector, in degrees.

LAT1
The latitude-like coordinate of the first vector, in degrees.

LON2
The longitude-like coordinate of the second vector, in degrees.

LAT2
The latitude-like coordinate of the second vector, in degrees.

This function returns in SEPARATION the separation angle, in degrees, between the two vectors (LON1, LAT1) and (LON2, LAT2).

BINPLOT

This procedure produces a log-log display of the supplied data.

Calling Sequence

BINPLOT, xedg, xwid, yval, yerr

[This procedure is written in IDL]

Arguments

XEDG
An array of lower edges for the x-axis (abcissa) bins.

XWID
An array of edge widths for the x-axis (abcissa) bins.

YVAL
An array of y-axis data values for the corresponding x-axis bins.

YERR
An array of y-axis uncertainties for the corresponding x-axis bins.

Keywords

/ll
If present and non-zero, /ll specifies that the plot is to be displayed as a linear-linear plot.

/lO
If present and non-zero, /lO specifies that the plot is to be displayed as a linear-log plot.

/Ol
If present and non-zero, /Ol specifies that the plot is to be displayed as a log-linear plot.

/OO
If present and non-zero, /OO specifies that the plot is to be displayed as a log-log plot.

/OPLOT
If present and non-zero, /OPLOT specifies that the plot is to be displayed using the current plotting environment **without** erasing the screen.

GRIDTHICK

An input parameter specifying the axis and line thickness to be used for the display. This argument has the same effect as the IDL system variables `!X.THICK` and `!Y.THICK`. The default value of GRIDTHICK is 0.

LIMIT

An input parameter specifying the significance, in sigma, below which a data point will be displayed as an upper limit. Data values below this significance limit will be displayed as 2-σ upper limits.

DISMODE

An input parameter specifying the data display mode. Supported values are:

0 – The data are displayed using the standard format.

1 – The data are displayed as a cross with bars.

$\geq$2 – The data are displayed as diamonds.

TICSIZE

An input parameter specifying the size of the bars used for DISMODE = 1, given in screen normal units. The default value of TICSIZE is $0.25 \times$ `!P.CHARSIZE`.

IDL Keywords Accepted

The following IDL keywords are accepted: `BACKGROUND, CHARSIZE, CHARTHICK, XCHARSIZE, YCHARSIZE, COLOR, FONT, LINESTYLE, NOERASE, POSITION, PSYM, XRANGE, YRANGE, XSTYLE, YSTYLE, XMARGIN, YMARGIN, SUBTITLE, SYMSIZE, THICK, XTHICK, YTHICK, XTICKLEN, YTICKLEN, TICKLEN, XTICKNAME, YTICKNAME, XTICKS, YTICKS, XTICKV, YTICKV, XTICK_GET, YTICK_GET, TITLE, XTITLE, YTITLE, XTYPE, YTYPE, YNOZERO`.
See the *IDL Reference Guide* for a description of the usage of these keywords.

Discussion

NOTE: IDL does not correctly handle log plots with ranges that are not more than a decade.

Related Routines

See the related procedures BINPLOT_LABEL and BINPLOT_SYMBOL.

BINPLOT_LABEL

This procedure produces plot labels on displays produced by BINPLOT.

Calling Sequence

BINPLOT_LABEL, x0, y0, text

[This procedure is written in IDL]

Arguments

X0
The x-position of the beginning of the plot label, specified in screen normal units.

Y0
The y-position of the beginning of the plot label, specified in screen normal units.

TEXT
A character string containing the text to be displayed.

IDL Keywords Accepted

The following IDL keywords are accepted: `CHARSIZE`.
See the *IDL Reference Guide* for a description of the usage of these keywords.

Related Routines

See the related procedures BINPLOT and BINPLOT_SYMBOL.

BINPLOT_SYMBOL

This procedure produces plot symbols for labeling displays created by BINPLOT.

Calling Sequence

BINPLOT_SYMBOL, xcent, ycent, bar_size

[This procedure is written in IDL]

Arguments

XCENT
The x-position for the center of the plot symbol, specified in screen normal units.

YCENT
The y-position for the center of the plot symbol, specified in screen normal units.

BAR_SIZE
An input parameter specifying the size of the symbol to be displayed, given in screen normal units. This argument may be a 2-element array, in which case the first element represents the symbol size in the x-direction and the second element represents the symbol size in the y-direction.

Keywords

DISMODE
An input parameter specifying the data display mode. Supported values are:

0 – The data are displayed using the standard format.

1 – The data are displayed as a cross with bars.

$\geq$2 – The data are displayed as diamonds.

SYMSIZE
An input parameter controlling the size of the "PSYM" symbols added to the data point symbol. Default is standard idl usage.

TICSIZE
An input parameter specifying the size of the bars used for DISMODE = 1, given in screen normal units. The default value of TICSIZE is $0.25 \times$ `!P.CHARSIZE`.

IDL Keywords Accepted

The following IDL keywords are accepted: `COLOR, LINESTYLE, PSYM, THICK.` See the *IDL Reference Guide* for a description of the usage of these keywords.

Discussion

When combined with the procedures BINPLOT and BINPLOT_LABEL, this procedure is useful when creating a legend for a display.

Related Routines

See the related procedures BINPLOT and BINPLOT_LABEL.

BLD_LC

This procedure automates the generation of a pulsar light curve from the OSSE pulsar data.

Calling Sequence

BLD_LC, targname, startdate, stopdate

Arguments

TARGNAME

A character string identifying the name of the target for which processing is to be performed. This name must be consistent with names in the OSSE Target Database and ephemeris file. It is used to find the target ID and ephemeris information. It can be abbreviated as long as the specification is unique. Examples: 'PSR0531+21', '0531'; 'CRAB' and 'VELA' are also accepted.

STARTDATE

A character string identifying the start time of the period to be analyzed. The form is 'yy/ddd hh:mm:ss' and can be abbreviated to any specification including at least 'yy/ddd'; any missing elements are assumed to be zero. STARTDATE and STOPDATE are used to identify command states which must be examined and to find the appropriate production SPM and PDF datafiles to process. Epoch folding is performed only for data between STARTDATE and STOPDATE.

STOPDATE

A character string identifying the stop time of the period to be analyzed. The form is 'yy/ddd hh:mm:ss' and can be abbreviated to any specification including at least 'yy/ddd'; any missing elements are assumed to be zero. A STOPDATE of 'yy/ddd' will process datafiles up through 'yy/(ddd-1) 24:00:00'.

Keywords

PROD_DIR

An input character string specifying an alternate directory for production data files. The default is '$PROD'.

OUT_DIR

An input character string specifying the directory for the storage of the output selection (PSF) and PDF file list. The default is the current directory.

SPM_PREFIX

An input character string specifying an alternate production product type. The default value is 'P' for standard production. Use this keyword to process quicklook ('Q') or realtime ('R') data files. The specified prefix is also applied to PDF file names.

SDBDIR

An input character string specifying the directory containing the _SPM.SDB files for the time interval of interest. The default is '$PROD'.

PDFDIR

An input character string specifying the directory containing the PDF files for the time interval of interest. The default is '$PROD'.

/NOSEL

If present and non-zero, /NOSEL indicates that no selection file will be produced and no data selection will be performed during epoch folding. The resulting livetimes will represent **clock** time only.

SEL_FILE

A character string specifying the name of a previously generated selection (PSF) file which will be used by BLD_LC. No selection file will be produced by BLD_LC.

NPHASBIN

The number of phase bins in the output light curve. The default value is 32.

DPTBL_SEL

A character string specifying the detector pointing table to use in processing the PDF data. Acceptable values are 'PRI' for primary and 'SEC' for secondary. If not specified, the value is derived from the command state data base.

EPHEM_FILE

An input character string specifying the path and filename for a pulsar ephemeris file. The format of this file is the same as Taylor's PSRTIME.DAT, including two header lines. This keyword is useful for processing pulsar data using ephemerides not included in the standard file.

/NOEFOLD

If present and non-zero, /NOEFOLD specifies that the EFOLD record will be set up but that EFOLD itself will not be run.

/QUIET

If present and non-zero, /QUIET specifies that the diagnostic output of EFOLD will be suppressed.

EFOLD_FILE

A character string specifying the name of the output EFOLD record file. If not specified, a default name constructed from the pulsar name will be used.

SOURCE_OFFSET

An input parameter specifying the scan offset, in degrees, around the specified target which should also be considered "source" pointing positions. The default value is 1.0 degrees. This value must be set large enough so that the pulsar of interest is within SOURCE_OFFSET degrees of at least one of the pointing positions. Note that data from all pointings in the specified pointing table which have pulsar data will be accumulated.

/ACC_OVRFLW

If present and non-zero, /ACC_OVRFLW specifies that RATE mode data records with flags indicating data overflow will be accepted and summed into the light curve. This is primarily useful where a known subset of the windows overflow (and can be discarded after the fact) while the remainder are safe (overflow very rarely or not at all).

/RADIO_PH0

If present and non-zero, /RADIO_PH0 indicates that the light curve will be shifted to place the phase of the radio pulse at phase 0 of the output light curve. The radio phase is defined in the standard ephemeris file and is stored in the EFOLD record in the variable EPH_T0GEO

Discussion

This procedure is the pulsar equivalent of BLD_QUAD_DIF and has a similar command structure. It epoch-folds the OSSE pulsar data according to ephemeris data stored in the format used by Taylor (PSRTIME.DAT). The ephemeris table is searched for the object name given as input; if an ephemeris entry valid over the specified interval is found, the ephemeris data are stored in the EFOLD record. Other relevant information is also stored. Unless keywords override this function, a selection file is produced from the associated SDB files with minimal data selection. A list file is produced for all associated PDF files. The resulting EFOLD records are written to a file (.EFD) and can be accessed using the routines EFOLD$OPEN, EFOLD$READ and EFOLD$READALL .

The command state database is used to find the appropriate strategy tables. Sets of command states with consistent instrument pulsar configurations are identified and one light curve (and one EFOLD record) is produced for each such configuration.

Examples of the use of BLD_LC may be found in the *OSSE Pulsar Analysis Cookbook*.

BLD_LOGBINS

This procedure creates logarithmically-spaced bins covering the specified range. The format of this command is:

Calling Sequence

BLD_LOGBINS, nbins, lower, upper, edges, widths

[This procedure is written in IDL]

Arguments

NBINS

A scalar integer specifying the number of bins to create. If keyword FIRSTWID is set, this is the number of bins created.

LOWER

A scalar specifying the lower bound of the range for which the binning is to be performed.

UPPER

A scalar specifying the upper bound of the range for which the binning is to be performed.

EDGES

An output array containing the lower edges of the bins covering the specified range.

WIDTHS

An output array containing the widths of the bins covering the specified range.

Keywords

FIRSTWID

If set, used to specify desired width of first channel. In this case, FIRSTWID, EMIN and EMAX are used to specify channels and NBIN is returned. NBIN is rounded down to the nearest integer, resulting in a first channel width that may be slightly larger than FIRSTWID .

Related Routines

See the related procedure BLD_SQRTBINS.

BLD_QUAD_DIF

This procedure performs quadratic background estimation and background subtraction for OSSE spectral memory data for the specified target and covering the specified time range. This is one of the primary analysis tools for the standard processing of OSSE spectral data. The format of this command is:

Calling Sequence

BLD_QUAD_DIF, targname, startdate, stopdate

[This procedure is written in IDL]

Arguments

TARGNAME

A character string specifying the name of the target for which the processing is to be performed. This name must be consistent with the name given in the OSSE Target Database; it may be abbreviated as long as the specification is unique.

STARTDATE

A character string specifying the start time of the interval over which the processing is to be performed. The form of STARTDATE is "yy/ddd hh:mm:ss". It may be abbreviated as long as the "yy/ddd" is specified; any missing elements are assumed to be zero.

STOPDATE

A character string specifying the stop time of the interval over which the processing is to be performed. The form of STOPDATE is "yy/ddd hh:mm:ss". It may be abbreviated as long as the "yy/ddd" is specified; any missing elements are assumed to be zero.

Keywords

/BACKGROUND

If present and non-zero, /BACKGROUND permits analysis of the background fields for a specified source. Each bkg position is used as source and the source position(s) are used for background.

/CAT_SAVE

If present and non-zero, /CAT_SAVE specifies that the IDL structures containing the relevant parameters from the SDB headers of the input SDB files are to be saved in IDL save files. These save files are stored in "OUT_DIR:<spm-input_name>.CAT".

/CONFIG

If present and non-zero, /CONFIG specifies that only the OSSE Command State

Database is to be analyzed and the relevant SPM datafiles are to be identified. No data will be read and no spectral differences will be performed.

DIF_FILES

Output character string array of full file name(s) (including directory and version number) of the resultant difference files.

DQHAZ_MASK

Input long specifying DQHAZ bits (in the SDB header) to ignore. Default is reject data if any DQHAZ bit is set (DQHAZ ne 0). eg. To ignore bit 0, set DQHAZ_MASK = 1.

DQINFO_MASK

Input long specifying DQINFO bits (in the SDB header) to use in data quality screening. Default is to ignore contents of DQINFO. A bit set in the mask will cause data rejection if the corresponding bit is set in a spectrum's DQINFO status. eg. To reject data with DQINFO bit 0 set, set DQINFO_MASK = 1.

DSFSTAT_MASK

An input long specifying DSFSTAT bits to use in data quality screening. Default is to ignore contents of DSFSTAT. A bit set in the mask will cause data rejection if the corresponding bit is set in a spectrum's dqInfo status. So, to reject data from ODC (bit 20), set DSFSTAT_MASK $=2L^{20}$.

EARTHSEP

Input float specifying the minimum earth separation (in degrees) for consideration as good data. Default is 72 degs - approximately the earth limb +50 km.

FILE_EXT

Character string specifying the file extension for the output difference spectral files. The default value is ".SDB".

FILE_MOD

Character string specifying the file modifier for the output difference spectral files. The default value is "_DIFF".

MIN_LTL

Input float specifying the minimum live time in the low range (in seconds) for consideration as good data. Default is 0. That is, no minimum livetime selection.

/NOPRI

An input logical flags, if present and nonzero will prevent analysis of the associated pointing table type, "PRI" when validating viewing configurations. See TABLE-TYPE keyword.

/NOSEC

An input logical flag, if present and nonzero will prevent analysis of the associated pointing table type, "SEC" when validating viewing configurations. See TABLE-TYPE keyword.

/NOSOL
An input logical flag, if set (nonzero) will prevent analysis of the associated pointing table type, "SOL" when validating viewing configurations. See TABLETYPE keyword.

/NOSTANDARD
An input logical flag, if present and nonzero will prevent standard environment screening from being performed.

/NOTRANS
An input logical flag, if present and nonzero will prevent analysis of the associated pointing table type "TRANS" when validating viewing configurations. See TABLETYPE keyword.

OUT_DIR
Character string specifying the directory for the output difference files. The default value is the current directory.

PROD_DIR
Character string specifying the directory for the input SPM files. The default value is "$PROD".

SOURCE_OFFSET
Input scalar value specifying the detector scan offset, in degrees, which should also be considered "source" observations. The default value of SOURCE_OFFSET is 1.

This argument is useful when analyzing observations having complex pointing tables (*e.g.*, the Galactic center) where multiple aspects to the source are used. Care should be taken when using this argument to ensure that it is not set larger than the typical background offset since, in that case, all pointings would be treated as "source" observations.

SPM_PREFIX
Character string used to specify that an alternate production type data should be processed. The default value is "P", indicating standard production mode data are to be processed. This argument may be set to "Q" to specify quick-look data or to "R" to specify real-time data products.

SPM_SUFFIX
An input character string, specifying alternate production SDB filetype. Default is "_SPM" for standard spectral memory sdb files.

TABLETYPE
An input string array of one or more positioning table type names. Valid names are "PRI", "SEC", "SOL" and "TRANS". If one or more positioning table types are specified, only those types will be examined. Default (without the keyword) is to examine all four types.

TARGDEF
Input IDL structure which specifies a target other than those found in the Plan-

ning Target Catalog. This structure is generally of the form of the target catalog structure, but only requires the following fields:

```
Name - character string name (12 chars max)
RA   - float, J2000 RA in degrees
Dec  - float, J2000 Dec in degrees
SrcID - long, Target Catalog ID number, best set to
        number less than 100 for targets not in OSSE
        catalog to avoid confusion.

example (using anonymous structures):
IGR>   TargDef = {Name:'James Bond', RA: 325.123,
       Dec: -54.32, SrcID: 007}
```

USERSCREEN
Character string specifying the name of a user-supplied IDL procedure to perform additional data screening prior to the AU definition. The default is that no additional user screening is performed (see below for a description of the standard data screening that is applied). A description of the requirements for the IDL procedure specified by USERSCREEN is given below.

Discussion

The procedure BLD_QUAD_DIF is one of the primary analysis tools for the standard processing of OSSE spectral data. This procedure searches the OSSE Command State Database over the interval specified by STARTDATE and STOPDATE for observations of the source specified by TARGNAME, identifying all of the relevant SPM data files. The directory containing the SPM files to be used can be specified using the keyword PROD_DIR. The data product to be used (Production, Quick-look or Real-time) can be specified using the keyword SPM_PREFIX.

The Command State Database is also used to create the AU definition for the specified target and the appropriate detector pointing strategy tables, using all detector positions within SOURCE_OFFSET of the target as "source" observations. Note that any detector position not considered a "source" observation may be used as a "background" accumulation. Also, it is assumed that the detector positioning table alternates between source and background observations (*e.g.*, that every other position is a "source" observation).

If the /CONFIG keyword is specified, processing is finished here; otherwise, the processing is continued as described below.

On a file-by-file basis, all of the data (source and background) associated with the specified target are read in and standard data screening is applied. The standard data screening consists of the following tests (see the "Spectral data Base Header Definition" document for a description of the SDB header parameters):

- EARTHSEP $> 72^\circ$ (*e.g.*, the OSSE detector is not looking below the earth's limb).

- DQHAZ = 0 (to exclude data for which the data quality is considered poor).

In addition to the data screening listed above, user-screening may also be applied by specifying the keyword USERSCREEN. The IDL procedure specified by USERSCREEN must be a procedure having two arguments, an SDR array and a returned array of indices into the input SDR array which satisfy the user-defined screening.

NOTE: The input SDR array will contain pointers to data from all four detectors (as appropriate for the specified target) and will be in chronological order. The array of returned indices should preserve the chronological order of the SDRs.

The data passing the above data screening are then used to create valid AU's using the nearest three or four background accumulations closest in time to the "source" observation. The maximum time difference between the "source" observation and any of the valid background accumulations making up the AU is 500 seconds. The minimum number of background accumulations required for a valid AU is three; if three valid background accumulations cannot be found, no AU is generated for that "source" observation.

Once all of the valid AU's have been identified, the background estimation and subtraction is performed. The background estimation is performed on a channel-by-channel basis by using a quadratic interpolation in time using all of the background spectra in the AU. The background estimate is then subtracted from the "source" spectrum and the resulting difference spectrum is written to an output SDB file. The name of the output difference file is based on the root of the input SDB file, with the "_SPM" modifier replaced by the string specified by FILE_MOD and the file extension specified by FILE_EXT. The directory in which the output difference file is written is specified by OUT_DIR.

This routine also provides a detailed listing on the screen of the valid command states during the specified interval, the AU definitions for each command state, the number of spectra passing data screening, and the number of spectra for which valid AU's were generated.

BLD_SQRTBINS

This procedure creates square-root-spaced bins covering the specified range. The format of this command is:

Calling Sequence

BLD_SQRTBINS, nbins, lower, upper, edges, widths

[This procedure is written in IDL]

Arguments

NBINS
A scalar integer specifying the number of bins to create.

LOWER
A scalar specifying the lower bound of the range for which the binning is to be performed.

UPPER
A scalar specifying the upper bound of the range for which the binning is to be performed.

EDGES
An output array containing the lower edges of the bins covering the specified range.

WIDTHS
An output array containing the widths of the bins covering the specified range.

Related Routines

See the related procedure BLD_LOGBINS.

BUILDMODEL

This procedure allows interactive selection of functions and function parameter values for modeling the incident photon spectrum.

Calling Sequence

BUILDMODEL, mdldef, [fdomain]

Arguments

MDLDEF

Output or input/output model definition record (or record array). The model record is an IDL record of structure MODSTR defining a photon model.

FDOMAIN

Input value specifying the source to which the model applies. FDOMAIN is an optional argument when analyzing a single source. When analyzing multiple sources, FDOMAIN must be specified for models for second or successive sources.

If present, the FDOMAIN argument must be either source name, target ID (from OSSE planning catalog), or structured record. If the input is a record, it must be either of structure type PLANCAT, or be a record of an anonymous structure containing the fields NAME (string), RA (float), DEC (float), and SRCID (long).

In all cases, the argument FDOMAIN is used to locate the source in the OSSE planning catalog. If a specified source name is not found, an error results. If a specified source target ID is not found, a warning is issued but execution continues.

To add functions for a second source to an existing array of model definition records, the keyword APPEND should be set.

Keywords

VERBOSE

Input switch which if present and nonzero, specifies that informational message are to be printed. By default, only error and warning messages are printed.

APPEND

Input switch which if present and nonzero, specifies that the new model function(s) should be appended to any existing functions defined in the argument MDLDEF. This keyword is used when adding model definitions for a second source to an existing model array. By default, the new model function(s) will replace the contents of the MDLDEF argument.

Discussion

This procedure allows interactive selection of functions and function parameter values for modeling the incident photon spectrum. It must be called once for each source to be modeled. (See keyword APPEND.) Functions supported are:

polynomial	exponential	gaussian
powerlaw	obsolete bpl use 25	thermal brem
brem w/ Gaunt fac	Wien	Sunyaev-Titarchuk
synchrotron	blackbody	compton reflected
comptonized brem	relativistic brem	pair plasma downsc
SN Co57 features	positronium	integrated gaussian
powerlaw * expon	expon w/ reflection	constant
linear	quadratic	compton scattered rad
pwrlaw w/ break	**User Model**	

Interactive help about the models and their parameters may be accessed at any prompt within BUILDMODEL by simply entering a question mark.

A brief description of each of these functions is provided in the *OSSE Data Analysis User's Guide*.

The *User Model* option allows for user-defined functions. User-defined functions must follow the format specified by the templates provided in the *OSSE Data Analysis User's Guide*.

Related Routines

See the related routines MODEL_READ, MODEL_WRITE, and MODELSTRUC.

BUILD_RMTRX

This procedure builds a compound OSSE intrument response matrix.

Calling Sequence

BUILD_RMTRX, sdr

Arguments

SDR

Input SDR pointer arrays or subarray containing one or more spectra for which to build a response. The energy edges and widths must be monotonically increasing vectors. To build a reponse matrix without the use of an SDR pointer array (i.e., if a dummy variable is put in its place), then keywords CHEDG, DETSTRUC, and CRC must be provided.

Keywords

CHEDG

Vector of energy channel lower edges of desired binning for columns in response matrix (i.e., channel space energies). This keyword should only be set if no SDR is provided as input. This vector must have consecutive positive values greater than zero.

CHWID

Vector of energy channel widths of desired binning for columns in response matrix (i.e., channel space energies). This keyword should only be set if no SDR is provided as input.

DETID

Scalar specifying ID of detector to build the response matrix. If this keyword is set, then the detid in the SDR will not be used. This keyword need only be set if an SDR pointer array is NOT provided as input to the program.

CRC

Input structured record containing the same fields as the IGR$CRC record, required only when SDR argument is a dummy.

DETSTRUC

Input structured record containing detector parameters, as returned by the FILL_DETSTRUC function. Required only when SDR argument is a dummy.

ERSTAT

Status of successful completion (0 - good, 1 - bad).

FULLNAME

string or string array defining full name of output response file(s). This should be used only if the user wants each spectrum's response to be outputted to individual files. The number of file names must equal nspec*nsrc. If both keywords OUTROOT and FULLNAME are set, OUTROOT will be ignored. When this keyword is not set, the default is to write the response matrix and other data to the common block FFOLD$COM.

PHEDG

A vector of user-specified photon lower energy edges. If this keyword is not specified then the procedure will produce photon edges for 460 channels using the utility BLD_LOGBINS, starting at 0.04 MeV.

PHWID

A vector of user-specified photon energy widths. If this keyword is not specified then the procedure will produce photon widths using previously generated photon energy edges.

OUTROOT

Output string name containing the ROOT NAME of each outputted response (one per each record in input SDR). Additional information will be appended to this to create the final file name. When this keyword is not set, the default is to write the response matrix and other data to the common block FFOLD$COM.

OUTSDB

The keyword OUTSDB if set to a string containing a filename, specifies that a new SDB file containing the compound response matrix as well as the original SDR data should be created. The argument string provides the name for the new SDB file. This is the preferred method of storing the response matrix.

VERBOSE

Input switch which, if present and non-zero, will display progress information as the compound response matrix is built.

Discussion

This procedure creates a two dimensional array defining a compound OSSE instrument response. A compound response matrix, as opposed to a simple response matrix, is the instrument response with the off-angle (i.e. background) responses to the target subtracted from the on-target response.

A compound response will be generated for each record in the input SDR array. These will be concatenated into a single matrix, which can then be used to fit multiple spectra simultaneously from a single model (e.g. when fitting four detectors).

Related Routines

See the related routines FFPREP_SPEC, DTW_CORRECT, and DO_FOLDANAL.

CNTS_TO_PHOTONS

This procedure generates an SDR containing OSSE spectral data in photon space units (i.e. units are $photons/sec\text{-}cm^2\text{-}MeV$).

Calling Sequence

CNTS_TO_PHOTONS, fsdr, modeldef, psdr

Arguments

FSDR

Input SDR pointer array or subarray containing count spectral data. This SDR should be in units of counts/sec-channel. If not, then the program will try to adjust the units.

MODELDEF

Input record array, of structure MODSTR, from which parameter data is obtained to generate a photon model. The photon model is used to calculate an efficiency factor described below.

PSDR

Output SDR containing photon space spectral data.

Keywords

MTXFILE

Input file name for response matrix. If not specified, the response matrix will be located using data contained in the FSDR.

ERSTAT

Status of successful completion (0 - good, 1 - bad).

Discussion

This procedure generates an SDR containing OSSE spectral data in photon space units (i.e. units are $photons/sec\text{-}cm^2\text{-}MeV$). An efficiency factor is obtained by dividing the model function in photon space with the model function in count space. This efficiency factor is then multiplied to the count spectral data to obtain an approximate value for the spectral data in photon space.

Related Routines

See the related routines MODEL_READ, MODEL_WRITE, DO_FOLDANAL, and PLOT_SRC.

COMPOUND_AREA

This procedure calculates the compound photopeak area at the given energies ENERGY using the src/bkg angles and weights (Aij's) in the input crc structures

Calling Sequence

COMPOUND_AREA, crc, energy, comparea

Arguments

CRC

Input array of idl compound response coefficients. CRC structures which identify a set of scan, xscan angles and their respective weights for creating compound response. CRCs may be as for an array of sdrs: if two-dimensional, will return summed over first dimension only.

ENERGY

Input float array containing 1 or more energies at which photopeak response is to be calculated.

COMPAREA

Output array of idl compound response areas (in cm^2), one for each input energy element. If CRC input is 2-d, then return array is (n_energy, 2nd dim) (eg. input: CRC(4,8),E(20) COMPAREA(20,8) summed over first dimension of the CRCs.)

Keywords

CRC_AREA

Specifies to return area by CRC. Array is (n_energy, n_crc).

AXISAREA

Specifies on-axis area.

CONFUSE

This procedure searches the OSSE Target Database to identify sources which might be confusing sources for the specified spacecraft attitude and target(s) of interest. The format of this command is:

Calling Sequence

CONFUSE, lun, att, cat, itargs, scanang, xscanang

[This procedure is written in IDL]

Arguments

LUN
An integer specifying the logical unit to which the display is to be written. If LUN < 0, the display will be written to the screen.

ATT
An input structure of type ATTSTRUC which defines the spacecraft attitude and observation interval for which the search for potentially confusing sources is to be performed.

CAT
An input structure array of type PLANCAT containing information about the potentially confusing sources. This is the list from which the potentially confusing sources are generated.

ITARGS
An input structure scalar or array of type PLANCAT containing the primary target(s) for which potentially confusing sources are to be generated.

SCANANG
An input angle, in degrees, specifying the scan region around the primary target(s) specified by ITARGS within which confusing targets are to be identified.

XSCANANG
An input angle, in degrees, specifying the trans-scan region around the primary target(s) specified by ITARGS within which confusing targets are to be identified.

Discussion

This procedure searches the target list specified by CAT for potentially confusing objects which lie within the SCANANG and XSCANANG of the primary target(s) specified by ITARGS. A listing of the potentially confusing sources is written to the logical unit specified by LUN.

NOTE: **This routine does not incorporate the details of the OSSE internal alignment and alignment to the spacecraft. This introduces errors generally less than** $0.1°$**.**

COPYREC

This procedure copies a single IGORE record element to another IGORE record element having the same structure. The format of this command is:

Calling Sequence

COPYREC, r_source, r_destination

Arguments

R_SOURCE

The input IGORE record pointer, or single element of a pointer array, for which a copy is to be made.

R_DESTINATION

The output IGORE record pointer, or single element of a pointer array, into which the input record is to be copied.

Keywords

NAME

A character string specifying the name to be used for the output IGORE record. Will be truncated if longer than 15 characters.

/UNIQUE

If present and non-zero, /UNIQUE forces the generation of a unique name for the output IGORE record. This keyword is ignored if NAME is specified.

Discussion

The input record R_SOURCE must exist prior to the application call. If the output record R_DESTINATION exists but does not have the same structure as the input record R_SOURCE, an error message is displayed and the application is aborted. If the output record R_DESTINATION does not exist, it is created. The name will be truncated, if it is longer, to 15 characters. If neither of the keywords are present, the name of the created record is taken from the name of the IDL variable passed as R_DESTINATION.

Copying a record element using COPYREC is different from merely copying the pointer since COPYREC actually copies all of the associated data from the input record to the output record. An example of the usage of COPYREC is given below:

```
Igore> SDR$CREATE, r_sdr1, 10
Igore> SDR$CREATE, r_sdr2, 2
```

```
Igore> COPYREC, r_sdr1(5), r_sdr2(0)
Igore> sdr2 = r_sdr1(5)
Igore> DELREC, r_sdr1
Igore> COPYREC, r_sdr2(0), r_sdr3
Igore> COPYREC, r_sdr2(0), r_tst1, name='r_sdr4'
Igore> COPYREC, r_sdr2(0), r_tst2, /unique
```

In this example, two SDR records are created; R_SDR1 containing ten elements and R_SDR2 containing two elements. The 5^{th} element of R_SDR1 is then copied to the 0^{th} element of R_SDR2. The pointer for the 5^{th} element of R_SDR1 is also copied to the IDL variable SDR2. The record R_SDR1 is then deleted. Note that the record element R_SDR2(0) can still be used to access the data which was originally in the 5^{th} element of R_SDR1, but SDR2 no longer contains a pointer to a valid record since the record to which it was pointing, R_SDR1, has been deleted. The 0^{th} element of R_SDR2 is then copied to the record R_SDR3, which is created by the procedure and has the name R_SDR3. The 0^{th} element of R_SDR2 is also copied to the record R_TST1, which is created by the procedure and has the name R_SDR4. Finally, the 0^{th} element of R_SDR2 is copied to the record R_TST2, which is created by the procedure and is given a unique name.

The procedure DUPREC should be used to copy an entire array of records.

The procedure MERGEREC can be used to copy a non-contiguous record array into a contiguous array.

CRC_ROUNDER

This routine takes an input list of compound response matrices (CRC) and merges them into an output array (OCRC) with similar scan/xscan entries as determined by resol (angle in degrees, eg. 0.2) summed into single output entry. Eg. weighted average of Aij. The format of this command is:

Calling Sequence

CRC_ROUNDER, crc, resol, ocrc

[This procedure is written in IDL]

Arguments

CRC
An input IDL structure array of CRCs.

RESOL
An input angular resolution in degrees for identifying unique configurations.

OCRC
Output IDL structure containing the resultant CRCs.

Keywords

/BYDET
An input logical which if present and nonzero specifies that preservation by detector should occur. SCAN, XSCAN and ANGLES are rounded as a group then saved by detector.

Discussion

Related Routines

Also see the related function SUMDET.

CRC_TRANSFORM

This routine converts compound response matrices (CRC) from current target to new target specification, returning them in a new array of crcs. The format of this command is:

Calling Sequence

CRC_TRANSFORM, targdef, crc, outcrc

[This procedure is written in IDL]

Arguments

TARGDEF

An input parameter of either target id, target structure or string specifying target name. If target id, it will be used to look up target in catalog. If target structure, it must have either (ra, dec) fields or (l2,b2) fields, and name field. In some applications, addition of SRCID field is also desired, although if not specified, it will be assigned a SRCID outside the range of the planning catalog. If string, it is used as name for lookup in the catalog.

CRC

An input IDL structure array of CRC information structure type CRC_STRUC attitude information in these structures is used to compute aspect information to specified target, TARGDEF, in the output crc array, OUTCRC. There is one element in OUTCRC for every element in the input, CRC.

OUTCRC

Output IDL structure array containing the resultant CRC information with modified target attitude information.

CREREC

This procedure creates an IGORE record of the specified type with the specified name and number of elements. The format of this command is:

Calling Sequence

CREREC, structure_type, r_pointers, nrec

Arguments

STRUCTURE_TYPE
A character string specifying the IGORE structure type of the record to be created.

R_POINTERS
The IDL variable into which the created IGORE record pointers will be placed.

NREC
A scalar integer representing the number of elements in the record array to be created.

Keywords

NAME
A character string specifying the name to be used for the IGORE output record. Will be truncated if longer than 15 characters.

/UNIQUE
If present and non-zero, /UNIQUE forces the generation of a unique name for the IGORE output record. This keyword is ignored if NAME is specified.

Discussion

This routine will create an IGORE record having a structure type specified by STRUCTURE_TYPE and the number of elements specified by NREC. Pointers to the IGORE record are returned in the variable R_POINTERS. If neither of the keywords are present, the name of the IGORE record created will be taken from the name of the IDL variable passed as R_POINTERS. The name will be truncated, if it is longer, to 15 characters.

The command SHOWSTRUC can be used to list the available IGORE structures and the command SHOWFLD can be used to list the fields defined for a specified IGORE structure.

IGORE does not currently support record arrays of more than one dimension.

CROSS_PROD

This procedure calculates the vector cross-product between two vectors. The format of this command is:

Calling Sequence

CROSS_PROD, lon1, lat1, lon2, lat2, lon3, lat3

[This procedure is written in IDL]

Arguments

LON1
The input longitude-like component, in degrees, of vector #1.

LAT1
The input latitude-like component, in degrees, of vector #1.

LON2
The input longitude-like component, in degrees, of vector #2.

LAT2
The input latitude-like component, in degrees, of vector #2.

LON3
The output longitude-like component, in degrees, of vector #3.

LAT3
The output latitude-like component, in degrees, of vector #3.

Discussion

This procedure performs the vector cross-product between the two input vectors specified by (LON1, LAT1) and (LON2, LAT2), returning the coordinates of the resultant vector in (LON3, LAT3). The returned vector is given by $\vec{V_3} = \vec{V_1} \times \vec{V_2}$.

DAYCNV

This procedure converts Julian dates to the corresponding Gregorian calendar dates. The format of this command is:

Calling Sequence

DAYCNV, jd, year, month, day, hour

[This procedure is written in IDL]

Arguments

JD
The Julian date to be converted.

YEAR
The returned Gregorian year corresponding to the specified JD.

MONTH
The returned Gregorian month corresponding to the specified JD.

DAY
The returned Gregorian day number corresponding to the specified JD.

HOUR
The returned Gregorian hour and fractional hours corresponding to the specified JD.

Discussion

This procedure converts the Julian date specified by JD into the corresponding Gregorian date given by YEAR, MONTH, DAY and HOUR. Array values of JD are supported, in which case the returned arguments will be arrays of the same length.

DAY_OF_YEAR

This function returns the day-of-year for the specified Gregorian calendar date. The format of this command is:

Calling Sequence

doy = DAY_OF_YEAR(year, month, day)

[This function is written in IDL]

Arguments

YEAR
The year, used for determining leap years.

MONTH
The month for which the day-of-year is to be determined.

DAY
The day of the month for which the day-of-year is to be determined.

Discussion

This function returns in DOY the day-of-year for the Gregorian date specified by YEAR, MONTH and DAY.

Note that no checking is performed for valid combinations of YEAR, MONTH and DAY.

DAY_SUMMER

This procedure performs spectral summations of individual difference spectra (*e.g.*, as produced by BLD_QUAD_DIF). This is one of the primary analysis tools for the standard processing of OSSE spectral data. The format of this command is:

Calling Sequence

DAY_SUMMER, infiles, outfile [, status]

[This procedure is written in IDL]

Arguments

INFILES
A character string or string array specifying the name(s) of the SDB difference files which are to be summed.

OUTFILE
A character string specifying the name of the output SDB file to which the summed spectra are to be written.

STATUS
An optional character string specifying the status with which OUTFILE is to be opened. Valid values of STATUS are "NEW" to create a new output file or "OLD" to append the summed spectra to an existing SDB file. The default value of STATUS is "NEW".

Keywords

/BYCMDSTATE
An input logical switch specifying, if set, that data from each unique CmdState should be sreened and summed separately. The default value breaks on igrparamDetCfg_id change.

CHAN_RANGE
An input 2-element array specifying the channel range to be used during standard screening. The default value of CHAN_RANGE = [14, 40]. This keyword is being replace with E_RANGE.

/COVAR
An input logical switch specifying that covariance of the resulting sums should be computed and stored in the output GOOD_SUMFILE. The default is for no covariance computations. This switch is functional only with the GOOD_SUMFILE keyword specified.

DQHAZ_MASK
An input long specifying dqHaz bits to ignore. The action is to reject data if any dqHaz bit is set (dqHaz nonzero).

DQINFO_MASK
An input long specifying dqInfo bits to use in data quality screening. The default action is to ignore contents of dqInfo. A bit set in the mask will cause data rejection if the corresponding bit is set in a spectrum's dqInfo status.

/DSA_SCREEN
An input logical switch specifying, if set, that data from each unique DSA should be screened separately.

DSFSTAT_MASK
An input long specifying DSFSTAT bits to use in data quality screening. Default is to ignore contents of DSFSTAT. A bit set in the mask will cause data rejection if the corresponding bit is set in a spectrum's dqInfo status. So, to reject data from ODC (bit 20), set DSFSTAT_MASK $=2L^{20}$.

EARTHSEP
An input floating point number specifying the minimum earth separation (in degrees) for consideration as good data. The default is 72 degs to approximately the earth limb + 50 km.

E_RANGE
An input 2-element array specifying the energy range to be used during standard screening of a difference spectrum. The default range of E_RANGE = [0.045, 0.25].

GOOD_SUMFILE
An optional character string specifying the name of an output SDB file to which the "good sums" are to be written. See below for a description of the "good sums" data.

/NODSA
An input logical switch specifying, if set, that default separation of sums by DSA (detector step angle) is disabled. All SDRs from a given detector for a day are summed.

/NOSTANDARD
If present and non-zero, /NOSTANDARD specifies that standard data screening is *not* to be performed. If the keyword USERSCREEN is also specified then user-defined data screening will be performed, otherwise no data screening will be performed. The default is to perform standard data screening.

/ORBIT
An input logical switch specifying, if set, that data is to be summed into orbital time sums. Approximately 16 sums per day per detector are created, synchronized to periods of continuous target visibility. Boundaries of sums are the centers of occultations. This switch overrides default daily sums.

NOTE: Orbital sums cannot bridge day boundaries. For correct operation, the file names require the nomimal "date root" format so that a filename sort will put result in a chronologically ordered list.

PROD_DIR
Input string specifying the directory containing the input files. This switch overrides the default input directory specification of "$PROD" for the input SPM files.

START_TIME
Input string specifying the start time for the sum(s), overriding the default action of starting with the first input SDR. The format is "yy/ddd hh:mm:ss".

STOP_TIME
Input string specifying the stop time for the sum(s), overriding the default action of stopping with the last input SDR. The format is "yy/ddd hh:mm:ss".

THRESH
An input scalar specifying the chisqr probability threshold ($0 < \text{thresh} < 1$) for rejecting a difference spectrum during standard screening. The default value of THRESH is 0.9 .

USERSCREEN
A character string specifying the name of a user-supplied IDL procedure to perform additional data screening prior to summing the difference spectra. The default is that no additional user screening is performed (see below for a description of the standard data screening that is applied). A description of the requirements for the IDL procedure specified by USERSCREEN is given below.

User-specified data screening is performed after standard screening or instead of standard screening, depending on the keyword /NOSTANDARD.

/VERBOSE
An input logical switch to turn on diagnostic/informational messages. Use this keyword to get crc printouts.

Discussion

The procedure DAY_SUMMER is one of the primary analysis tools for the standard processing of OSSE spectral data. This procedure is used to sum individual difference spectra (*e.g.*, produced by BLD_QUAD_DIF) for each of the input difference files specified by INFILES, writing the summed spectra to the file specified by OUTFILE. Spectra from each detector and each target offset are treated separately.

On a file-by-file basis, all of the difference spectra are read in. If specified (*e.g.*, the /NOSTANDARD keyword is not specified), standard data screening is applied for each detector and target offset separately. The standard data screening consists of generating an "average" difference spectrum and comparing each of the individual difference spectra to the average. The comparison consists of two steps. First, the window rate for each of the individual difference spectra within the channel range specified by CHAN_RANGE is compared to the window rate for the "average" difference spectrum. Individual difference spectra for which the window rate has a probability exceeding that specified by THRESH are removed. Second, an "average" difference spectra is again generated, this time only using spectra passing the first selection criterion. Again, for each of the individual difference spectra, the spectrum within the channel range specified by CHAN_RANGE is compared with the "average" spectrum over the same channel range and the probability that the individual difference spectrum came from the same parent population is estimated. Individual difference spectra for which the probability that they came from the same parent population as the "average" exceeds the level specified by THRESH are removed. The spectra passing both of these data selection criteria are then summed and written to the file specified by OUTFILE.

In addition to the data screening listed above, user-screening may also be applied by specifying the keyword USERSCREEN. The IDL procedure specified by USERSCREEN must be a procedure having two arguments, an SDR array and a returned array of indices into the input SDR array which satisfy the user-defined screening.

NOTE: The input SDR array will contain pointers to data from all four detectors (as appropriate for the specified target) and will be in chronological order. The array of returned indices should preserve the chronological order of the SDRs.

NOTE: Since the user-defined screening routine is called using the IDL `CALL_PROCEDURE` mechanism, it *must be* compiled before the call to DAY_SUMMER. It *will not* automatically compile.

The user-defined screening is performed after the standard screening or, if the keyword /NOSTANDARD is specified, instead of the standard screening. If /NOSTANDARD is specified but USERSCREEN is not, then no data screening will be applied to the data.

The spectral uncertainties for the summed spectra written to the SDB file OUTFILE are based on the standard propagation of errors for the individual difference spectra. Since many of the original background spectra may have been used for the background estimation for more than one source spectrum, however, a correlation exists between individual difference spectra which shared background spectra and the standard error propagation will not properly estimate the uncertainties in the resulting summed difference spectrum. The proper error propagation, including the correlation terms, is performed if the keyword GOOD_SUMFILE is specified. In this case, the same data screening and spectral summation is performed, but the spectral uncertainties are estimated based on the original source and background used to create each of the individual difference spectra. This error estimation requires accessing the original SPM files from which the differences were created. These SPM files are assumed to be in the directory pointed to by the logical "$PROD". The resulting "good sums", containing the correctly estimated spectral uncertainties, are written to the output SDB file specified by GOOD_SUMFILE. Note that, since this process requires accessing each of the original spectra used to create the individual difference spectra, it can be time consuming.

There will be one output SDB record written to OUTFILE (and GOOD_SUMFILE, if specified) for each detector and target offset combination for each of the input SDB files specified by INFILES. This routine also provides a detailed listing of the number of spectra passing data screening and the number of detectors and target offset values that are processed.

DELREC

This procedure deletes an IGORE record or multiple IGORE records. The format of this command is:

Calling Sequence

DELREC [, r_pointers]

Arguments

R_POINTERS

An IDL variable containing pointers to the IGORE record(s) to be deleted. This argument is optional if the keyword NAME is used.

Keywords

NAME

A character string specifying the name of the IGORE record to be deleted. Will be truncated if longer than 15 characters.

/ALL

If present and non-zero, /ALL forces the deletion of **all** IGORE records referenced in R_POINTERS.

Discussion

This routine deletes an IGORE record specified either by the pointers contained in the IDL variable R_POINTERS or by the name of the IGORE record using the NAME keyword. The name will be assumed to be the first 15 characters if it is longer. All VM associated with the record is freed. If R_POINTERS does not contain sequential pointers to the entire record, the record will not be deleted unless the /ALL keyword is present. If the /ALL keyword is present, **all** IGORE records pointed to by R_POINTERS are deleted. The /ALL keyword should be used with caution since the original IDL variables containing the record pointers will still be present but will no longer allow access to the IGORE records.

It is not possible to delete a single record element from an array of records. The routine MERGEREC may be used to copy various non-contiguous record elements into a contiguous record so that the original can be deleted. An example of the usage of DELREC is given below:

```
Igore> SDR$CREATE, r_sdr1, 10
Igore> SDR$CREATE, r_sdr2, 10
Igore> SDR$CREATE, r_sdr3, 10
Igore> SDR$CREATE, r_sdr4, 10
```

```
Igore> DELREC, r_sdr1
Igore> DELREC, name='r_sdr2'
Igore> index = [1, 3, 5, 7, 9]
Igore> MERGEREC, [r_sdr3(index), r_sdr4(index)], r_sdr5
Igore> sdr5 = [r_sdr3(index), r_sdr4(index)]
Igore> DELREC, sdr5, /all
```

In the above example four SDR records are created, each having ten elements. The record arrays R_SDR1 and R_SDR2 are then deleted. Next, the odd elements of the record arrays R_SDR3 and R_SDR4 are merged into a single record array R_SDR5. The pointers for the same odd elements from record arrays R_SDR3 and R_SDR4 are also copied into the IDL variable SDR5. Finally, the /ALL keyword is used with DELREC to delete all of the records referenced in SDR5. Note that this command deletes the record arrays R_SDR3 and R_SDR4; however, the original IDL variables containing pointers to these records still exist but no longer point to valid records. The record array R_SDR5 still contains the information from records R_SDR3 and R_SDR4 since MERGEREC copied the selected records (not just the pointers) into R_SDR5.

DISTINCT

This function returns the distinct elements of the input array. The format of this command is:

Calling Sequence

unique = DISTINCT(array)

[This function is written in IDL]

Arguments

ARRAY

An array for which the distinct elements are to be determined. Any variable type is supported.

Discussion

This function returns in UNIQUE the unique values of the input array ARRAY. Note that the returned array UNIQUE is unsorted.

DO_FOLDANAL

This procedure provides the primary user interface for the modeling, forward folding routines and fitting OSSE spectral data.

Calling Sequence

DO_FOLDANAL, sdr, mdldef

Arguments

SDR

Input SDR pointer array or subarray containing OSSE spectrum data. The SDR data should be in units of *counts/sec-channel*. If not, an attempt is made to convert the SDR data to the required units.

MDLDEF

Input/output model definition record (or record array). MDLDEF must be an IDL record of structure MODSTR. If the keyword switch NEWMOD is present and nonzero, MDLDEF is ignored on input. If MDLDEF is not initially defined, and no file name is passed via keyword argument MODELFILE, the utility BUILDMODEL is called. MDLDEF is updated with the best-fit model parameter values found by SUPERFIT.

Keywords

ERSTAT

Status of successful completion (0 - good, 1 - bad).

MTRXFILE

Optional string specifying input response matrix file(s). By default, the input SDR is checked for the presence of a response matrix in the parent SDB file or for an RELSPEC AUX field locating the correct response matrix. If multiple spectra are present in the SDR, MTRXFILE must be a string array containing the names of the corresponding response files to each spectrum. This keyword is provided only for compatiblity with previous versions.

LOGIT

Input switch which, if present and non-zero, specifies all inputs required for a subsequent run of DO_FOLDANAL are written to the log file FFOLDLOG.PRO.

ERANGE

Input two-element vector containing the minimum and maximum energy values specifying a subrange of energies for fitting.

MODELFILE
Input string containing name of file from which to read input model definition(s).

NEWMOD
Input switch which, if present and non-zero, specifies that interactive utility BUILDMODEL be called to generate new model definition(s).

INITMOD
Input switch which, if present and non-zero, specifies that initial parameter values from MDLDEF be used in fit instead of possible left-over best-fit values from a previous run.

SUPERFILE
File name for SUPERFIT's output log.

CHI2_INC
Input Real value specifying the $\Delta\chi^2$ value for SUPERDUPERFIT. This argument should be set when one or more parameters in the model definition will be evaluated by SUPERDUPERFIT. If not present, a default value for CHI2_INC is calculated based on a probability value of .68296 (1σ).

CHI2_SIGMA
Input scalar value defining the sigma value at which to evaluate $\Delta\chi^2$ for SUPERDUPERFIT. This argument is only used when at least one parameter in the model definition will be evaluated by SUPERDUPERFIT.

NPRINT
Input integer value specifying the desired log output from SUPERFIT. This log, if selected, reports fit information and statistics. It is distinct from the log controlled by the LOGIT keyword. If not present, or set to zero, no output is logged from SUPERFIT. If NPRINT is greater than zero, full output logging is enabled. If NPRINT is less than zero, output logging is selective enabled. NPRINT is then considered as a string of independent decimal digits. The position of each digit is not significant, but each digit value enables a specific output function. These are:

Value	Output selected
1 –	page eject between fits
2 –	initial condition
3 –	iteration values
4 –	intermediate param vals
5 –	convergence stop code
6 –	intermediate page eject
7 –	final fit statistics
8 –	final parameter values

9 – NY model values and difs

EPS1

User-specified value for SUPERFIT's convergence criteria. By default, the value is set to 0.001.

TARGNAMES

The keyword TARGNAMES is an optional input string array containing the name of each source for which fitting should be performed. By default, fits are attempted for all targets listed in the input SDR's CRC AUX records.

ADDSDR

The keyword ADDSDR an optional input SDR (array) containing the relevant data associated with spectra from other instruments. Units of data should be $\gamma/s - cm^2$. The required SDR fields are DAT (photon spectrum), ERR (uncertainties), EDG (photon energy edges), and WID (photon energy widths).

Discussion

This procedure provides the primary user interface for the modeling, forward folding and fitting of OSSE spectral data. It controls data and response matrix access and manipulation as well as the forward folding process itself.

In its simplest form, a call to DO_FOLDANAL requires only an SDR and an input MDLDEF argument. The latter may be undefined, if so, a model will be defined interactively. In most cases the response matrix (or matrices) can be accessed based on data in the input SDR, if not the MTRXFILE keyword must be used.

DO_FOLDANAL uses a generalized least-squares fitting routine known as SUPERFIT and optionally, depending on the contents of the input MDLDEF, SUPERDUPERFIT. Several keywords are simply passed along to these routines to control fitting and parameter evaluation.

Several levels of logging are available, controlled by the LOGIT and NPRINT keywords.

Data from instruments other than OSSE can be analyzed simultaneously with OSSE data via the ADDSDR keyword.

See the *OSSE Data Analysis User's Guide* for a detailed "cookbook" approach to analyzing OSSE data.

Related Routines

See the related routines BUILD_RMTRX, BUILDMODEL, MODEL_READ, MODEL_WRITE, MODELSTRUC, DTW_CORRECT, and FFPREP_SPEC for information on preparing to run DO_FOLDANAL.

For information about processing following DO_FOLDANAL, see the related routines CNTS_TO_PHOTONS and PLOT_SRC.

DTW_CORRECT

This procedure corrects each spectrum data record in the input SDR, and its associated uncertainties, with a DTW (Digital Time Window) efficiency factor.

Calling Sequence

DTW_CORRECT, sdr, [dtwarray]

Arguments

SDR

Input SDR array containing spectra to be corrected. The DTW-corrected spectra are replaced in this SDR, overwriting previous spectral data.

DTWARRAY

Optional DTW efficiency matrix with which to correct the spectral data. The array must have the format FLTARR(544,2,4,[2]), where the last dimension defines either X1 or X2 relative gain, respectively. See the *OSSE Data Analysis User's Guide* for information on generating a DTW efficiency array. This parameter overrides the keyword DTWFILE if both are present.

By default, the DTW efficiency array is read from either IGR$DAT:DEFLT_DTW_-RPHADIF_X1_GAIN.DAT or IGR$DAT:DEFLT_DTW_RPHADIF_X2_GAIN.DAT, depending on the instrument gain factor.

Keywords

DTWFILE

Optional input string containing name of the DTW file from which to obtain the DTW efficiencies. This file should contain a properly formatted 3 or 4-dimensional DTW array. Ignored if DTWARRAY is passed.

STATUS

Status of successful completion (0 - good, 1 - bad).

Discussion

This procedure corrects each spectrum data record in the input SDR, and its associated uncertainties, with a DTW (Digital Time Window) efficiency factor.

This procedure requires that the format of the input spectrum be exactly 544 channels. Therefore, if the OSSE spectrum is less than 544 channels (e.g., pulsar data), the DTW array must be properly resampled before calling DTW_CORRECT.

Related Routines

See the related routines FFPREP_SPEC, BUILD_RMTRX, and DO_FOLDANAL.

DUPREC

This procedure duplicates an IGORE record array. The format of this command is:

Calling Sequence

DUPREC [, r_source], r_destination

Arguments

R_SOURCE

An IDL variable containing the record pointers to the IGORE record to be duplicated. This argument is not used if the keyword SRC_NAM is provided.

R_DESTINATION

The IDL variable containing the record pointers for the output record array.

Keywords

/MERGE

If present and non-zero, /MERGE indicates that, rather than duplicating an entire input SDR array, DUPREC should duplicate only those elements contained in the input pointer array. The pointers may refer to multiple SDR arrays. /MERGE and SRC_NAM may not be combined.

NEW_NAM

A character string specifying the name for the IGORE output record. Will be truncated if longer than 15 characters.

SRC_NAM

A character string specifying the name for the IGORE input record. Will be assumed to be no longer than 15 characters.

/UNIQUE

If present and non-zero, /UNIQUE forces the generation of a unique name for the IGORE output record. This keyword is ignored if the NEW_NAM keyword is present.

Discussion

This routine duplicates an IGORE record array, including all of the associated indirect fields and records. The name of the IGORE input record is taken from the first valid pointer in R_SOURCE, unless the keyword SRC_NAM is provided. The name of the IGORE output record is taken from the name of the IDL variable passed as R_DESTINATION, unless one of the keywords NEW_NAM or /UNIQUE are provided. An example of the usage of DUPREC is given below:

```
Igore> SDR$CREATE, r_sdr1, 10
Igore> SDR$CREATE, r_sdr2, 10
Igore> DUPREC, r_sdr1, r_sdr3
Igore> DUPREC,/UNIQUE, r_sdr1, tst4

Igore> DUPREC, [r_sdr1, r_sdr2], r_sdr6
Igore> DUPREC,/MERGE, [r_sdr1, r_sdr2], r_sdr7
```

In the above example two SDR records are created, each having ten elements. The record array R_SDR1 is then duplicated, with the record name of the copy being R_SDR3 and the record pointers being stored in the IDL variable R_SDR3. Next, the record array R_SDR1 is again duplicated, the resulting record pointers being stored in the IDL variable TST4 and a unique output record name generated automatically.

Next, an incorrect attempt at duplicating the combination of record arrays R_SDR1 and R_SDR2 is made. In this example, only the record array R_SDR1 is duplicated and stored in the record array R_SDR6. The proper method for generating a copy of the combination of two or more record arrays is to use the keyword /MERGE as shown in the final example.

EDT

This procedure provides the ability to enter the EDT editor directly from the active IGORE session. The format of this command is:

Calling Sequence

EDT [, file_name]

[This procedure is written in IDL]

Arguments

FILE_NAME
An optional character string specifying the name of the file to edit.

Discussion

This procedure provides access to the EDT editor directly from the active IGORE session; using this command is significantly faster than spawning a process to execute the editor. The name of the file to edit is prompted for if the optional argument FILE_NAME is not provided.

EFOLD$OPEN

This procedure provides a method for opening an EFOLD file for subsequent access by IGORE applications. The format of this command is:

Calling Sequence

EFOLD$OPEN, lun, file_name

Arguments

LUN
An integer specifying the logical unit number to which the EFOLD file is to be attached.

FILE_NAME
A character string specifying the name of the EFOLD file to be opened.The length of this parameter cannot exceed 20 characters. This argument may represent a logical pointing to the EFOLD file to be opened. By default EFOLD$OPEN opens (an existing) file with readonly access.

Keywords

/CLOSE
If present and non-zero, /CLOSE specifies that the logical unit number or file is to be closed, if open, and re-opened using the specified logical unit number.

ERSTAT
A variable to receive the status of the open. A non-zero status code represents an an errror opening the EFOLD file. Zero indicates success.

/GET_LUN
If present and non-zero, /GET_LUN specifies that the logical unit number to be opened is to be obtained from the logical unit table and returned in LUN.

/NEW
If present and non-zero, /NEW indicates that a new EFOLD file is to be opened for write access.

/OLD
If present and non-zero, /OLD indicates that an existing EFOLD file is to be opened. Readonly access is the default for files opened with the /OLD keyword.

/WRITE
If present and non-zero, /WRITE specifies that the EFOLD file is to be opened with

read/write access. /NEW is the default for files opened with the /WRITE keyword.

Discussion

This routine provides a method for opening an EFOLD file for subsequent access by IGORE applications. For existing files, the search for the EFOLD file specified by FILE_NAME is performed as described in Chapter 2.

If the logical unit specified by LUN is already attached to a file and the keyword /CLOSE is not specified, an error is issued and the application aborted.

Note that the argument FILE_NAME cannot exceed 20 characters; for file names longer than 20 characters a logical should be used.

The default is to only open the EFOLD file SHARED for readonly.

Related Routines

See the related routine SELECT_FILES.

EFOLD$READ

This procedure provides a method of reading selected data from an EFOLD file into an IGORE EFOLD record or record array. The format of this command is:

Calling Sequence

EFOLD$READ, lun, r_efold, tag

Arguments

LUN
An integer specifying the logical unit number from which the data are to be read.

R_EFOLD
An EFOLD pointer array or subarray into which the data are to be placed.

TAG
A longword scalar or array specifying the tagwords to be read. For non-positive values of TAG, a sequential read is performed and the tagword of the record read is returned in TAG. Note that if a subarray is passed as TAG, sequential reads will be performed but the non-positive elements of TAG will *not* be modified to represent the actual tagwords read since IDL passes subarrays as temporary arrays.

Discussion

The EFOLD file must be opened (*e.g.*, using EFOLD$OPEN) prior to using this application.

If the number of array elements in the arrays R_EFOLD and TAG are different, a warning is issued and only the first N elements are read, where N is the minimum number of elements in the R_EFOLD and TAG arrays.

If a tagword specified by TAG does not exist, a warning is issued and the execution continued. The element of TAG producing the warning is set to zero and the data associated with the corresponding element of R_EFOLD are deleted.

EFOLD$READALL

This procedure provides a method of reading all the spectral data from an EFOLD file into an IGORE EFOLD record. The format of this command is:

Calling Sequence

EFOLD$READALL, lun, r_efold

Arguments

LUN
An integer specifying the logical unit number from which the data are to be read.

R_EFOLD
An output-only EFOLD record into which the data are to be placed.

Discussion

The EFOLD file must be opened (*e.g.* using EFOLD$OPEN) prior to using this application.

EFOLD$WRITE

This procedure provides the ability to write the information stored in an EFOLD record to a selected tagword of an EFOLD file. The format of this command is:

Calling Sequence

EFOLD$WRITE, lun, r_efold, tag

Arguments

LUN

An integer specifying the logical unit number of the EFOLD file to which the data are to be written.

R_EFOLD

An EFOLD pointer array or subarray from which the data are to be obtained.

TAG

A longword scalar or array specifying the tagwords of the EFOLD records to be written. For non-positive values of TAG, a sequential write is performed and the tagword of the record written is returned in TAG. Note that if a subarray is passed as TAG, sequential writes will be performed but the non-positive TAG elements will *not* be modified to represent the actual tagwords written since IDL passes subarrays as temporary arrays.

Discussion

The EFOLD file must be opened using EFOLD$OPEN with write access prior to using this application.

If the number of array elements in the arrays R_EFOLD and TAG are different, a warning is issued and only the first N elements are written, where N is the minimum number of elements in the R_EFOLD and TAG arrays.

If a tagword specified by TAG already exists in the EFOLD file, the data in the file for the corresponding tagword are overwritten.

EFOLD

This application epoch folds pulsar data contained in OSSE PDF format files. The format of this command is:

Calling Sequence

EFOLD, r_efold

Arguments

R_EFOLD
An input EFOLD pointer, or single element of a pointer array, on which EFOLD is to be run.

Discussion

The record R_EFOLD must be initialized prior to running EFOLD. The fields in R_EFOLD which must be initialized include the pulsar position, the pulsar ephemeris parameters, the PDF file name, and the number of phase bins to be produced. The procedure BLD_LC can be used to automatically initialize an EFOLD record.

A detailed description of the IGR$EFOLD record and the operation of EFOLD may be found in the Pulsar Analysis Cookbook.

EFOLD_PERIOD

This function calculates the period of a pulsar at a specified time using the ephemeris information stored in an EFOLD record. The format of this command is:

Calling Sequence

period = EFOLD_PERIOD(r_efold, tjd)

[This function is written in IDL]

Arguments

R_EFOLD
An input EFOLD pointer, or pointer array element, containing the pulsar ephemeris information.

TJD
The epoch, in truncated Julian days, at which the period is to be determined.

Discussion

This function returns the calculated pulsar period, in seconds, at the specified epoch, using the ephemeris information contained in the record R_EFOLD.

ENERGY_CAL

This function returns the channel edge energies for the specified OSSE detector and gain. The format of this command is:

Calling Sequence

edge_energies = ENERGY_CAL(ndet, gain_ratio [, fwhm_ratio])

[This function is written in IDL]

Arguments

NDET

An integer value specifying the OSSE detector number (1–4) for which the channel edge energies are to be calculated.

GAIN_RATIO

A real value specifying the detector gain ratio (relative to nominal gain).

FWHM_RATIO

An optional real value specifying the detector resolution (FWHM) relative to the ground calibration. Typical values are in the range 1.017–1.033.

Keywords

CO60

A real value specifying the peak channel number of the 1.33 MeV line from the internal ^{60}Co spectrum. If this keyword is present, its value overrides the specification of GAIN_RATIO.

WIDTH

The name of an IDL variable which, if present, upon return contains the channel widths (in MeV) for the specified detector and gain.

FWHM

The name of an IDL variable which, if present, upon return contains the instrumental resolution FWHM (in MeV) for the specified detector and gain.

FWCO60

A real value specifying the FWHM, in channels, of the 1.33 MeV line from the internal ^{60}Co spectrum. If this keyword is present, its value overrides the specification of FWHM_RATIO.

/NOHIGH

If present and non-zero, /NOHIGH indicates that the returned values do not include the HIGH spectral memory range (*e.g.*, only the LOW and MEDIUM spectral memory ranges). The default to to return the LOW, MEDIUM and HIGH range values.

/NOMED

If present and non-zero, /NOMED indicates that the returned values do not include the MEDIUM or HIGH spectral memory ranges (*e.g.*, only the LOW spectral memory range). The default to to return the LOW, MEDIUM and HIGH range values.

/NODIFLIN

If present and non-zero, /NODIFLIN indicates that the returned values do not include differential non-linearity corrections. The default is to include differential non-linearity corrections.

TEMP

A real value specifying the detector temperature, in degrees Celsius, for which the differential non-linearity corrections are to be calculated. The default value is 21°C.

VERSION

Output string containing ENERGY_CAL version information.

Discussion

This function returns in EDGE_ENERGIES the channel lower edge energies, in MeV, for the specified detector and gain. The detector for which the channel edge energies is requested is specified by the argument NDET. The argument GAIN_RATIO represents the ratio of the detector gain to its nominal gain ($\sim$ 6 keV/channel in the LOW range). For $\times 2$ gain ($\sim$ 3 keV/channel in the LOW range) the value of GAIN_RATIO is 2. The detector gain information can be determined from the SDB header parameter STDRGAIN. The detector gain can also be specified using the keyword CO60.

The channel widths and instrumental resolution may also be returned using the keywords WIDTH and FWHM, respectively. By default, the returned array, EDGE_ENERGIES, will consist of 544 elements; 256 for the LOW spectral memory range, 256 for the MEDIUM spectral memory range and 32 for the HIGH spectral memory range. If the /NOHIGH keyword is present, the HIGH spectral memory range values are not returned and the arrays will consist of 512 elements (*e.g.*, LOW and MEDIUM ranges only). If the /NOMED keyword is present, the HIGH and MEDIUM spectral memory range values will not be returned and the arrays will consist of 256 elements (*e.g.*, LOW range only).

NOTE: At this time, OSSE HIGH range calibration is only approximate. More detailed calibrations will be included at a later date.

ERF

This function returns the error function for the specified value. The format of this command is:

Calling Sequence

return = ERF(value)

[This function is written in IDL]

Arguments

VALUE
The value for which the error function is to be evaluated. Both scalar and array types are supported.

This function returns in RETURN the error function evaluated at VALUE. This function uses the relationship between the error function and the complementary error function: $ERF = 1 - ERFCC$.

ERFCC

This function returns the complementary error function for the specified value. The format of this command is:

Calling Sequence

return = ERFCC(value)

[This function is written in IDL]

Arguments

VALUE

The value for which the complementary error function is to be evaluated. Both scalar and array types are supported.

Discussion

This function returns in RETURN the complementary error function evaluated at VALUE. This function is reported to have a fractional error which is everywhere less than 1.2×10^{-7}. The form of this function was taken from *Numerical Recipes*, Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vettering, William, Cambridge University Press, 1986, p. 164.

ERRLOG

This procedure is used to specify the IGORE Condition Handler logging mode. The format of this command is:

Calling Sequence

ERRLOG

[This procedure is written in IDL]

Keywords

/OFF

If present and non-zero, /OFF specifies that all error reporting from the IGORE Condition Handler is to be disabled.

Discussion

This routine provides user control of the IGORE Condition Handler logging of errors. Errors are reported to the screen and journal file (if active), or turned completely off. If the keyword /OFF is present, error logging is completely disabled; otherwise, error logging to both the screen and the journal file (if active) is enabled. Logging to a journal file can be initiated using the IDL command JOURNAL.

EULER

This procedure provides a method for transforming between galactic, celestial and ecliptic coordinates. The format of this command is:

Calling Sequence

EULER, lonin, latin, lonout, latout [, select]

[This procedure is written in IDL]

Arguments

LONIN

The input longitude-like component, in degrees, of the input vector to be transformed. Scalar or array data types are supported.

LATIN

The input latitude-like component, in degrees, of the input vector to be transformed. Scalar or array data types are supported.

LONOUT

The output longitude-like component, in degrees, of the transformed output vector.

LATOUT

The output latitude-like component, in degrees, of the transformed output vector.

SELECT

An optional integer parameter specifying the type of coordinate transformation to be performed. Supported values of SELECT are:

1 – Convert from celestial (1950) to galactic (ii).

2 – Convert from galactic (ii) to celestial (1950).

3 – Convert from celestial (1950) to ecliptic.

4 – Convert from ecliptic to celestial (1950).

5 – Convert from ecliptic to galactic (ii).

6 – Convert from galactic (ii) to ecliptic.

7 – Convert from celestial (2000) to galactic (ii).

8 – Convert from galactic (ii) to celestial (2000).

If SELECT is not provided, the user will be prompted for the transformation to be performed.

EXTRACT_COV

This procedure will extract the IGR$COVAR aux information from an sdr array returning it in an idl structure. The format of the command is:

Calling Sequence

EXTRACT_COV, sdr, cov

[This procedure is written in IDL]

Arguments

SDR
An array of SDR records containing the aux type IGR$COVAR.

COV
A two-dimensional (# offsets by # sdrs) array of extracted covariance information.

Keywords

ERSTAT
A output array containing the EXTRACT_COV status. Array elements are non-zero if the IGR$COVAR for an associated SDR record was not available.

EXTRACT_CRC

This procedure will extract the IGR$CRC aux information from an sdr array returning it in an idl structure. The format of the command is:

Calling Sequence

EXTRACT_CRC, sdr, crc

[This procedure is written in IDL]

Arguments

SDR
An array of SDR records containing the aux type IGR$CRC.

CRC
A two-dimensional (# offsets by # sdrs) array of extracted crc information.

Keywords

ERSTAT
A output array containing the EXTRACT_CRC status. Array elements are non-zero if the IGR$CRC for an associated SDR record was not available.

FDB$CLOSE

This procedure closes an FDB file previously opened using FDB$OPEN. The format of this command is:

Calling Sequence

FDB$CLOSE, lun

Arguments

LUN
An integer parameter specifying the logical unit number to close.

Discussion

This routine closes the logical unit specified by LUN. This routine also frees the appropriate logical units from the logical unit table, where necessary. Note that this routine will *not* close files opened by IDL.

The preferred method for closing FDB files is using the command IGR$CLOSE.

FDB$DIR

This procedure provides a method for performing a directory of records contained in a specified FDB file. The format of this command is:

Calling Sequence

FDB$DIR, lun

Arguments

LUN
An integer specifying the logical unit number of the file for which the directory is to be performed.

Discussion

The file must have been opened using FDB$OPEN prior to calling this routine. This routine also provides the ability to write the file directory to an ASCII file.

FDB$LOOK

This procedure provides a method for displaying the contents of a selected record contained in an FDB file. The format of this command is:

Calling Sequence

FDB$LOOK, lun

Arguments

LUN
An integer specifying the logical unit number of the file for which the contents listing is to be performed.

Discussion

The file must have been opened using FDB$OPEN prior to calling this routine. This routine prompts the user for the record to be listed and provides the ability to write this listing to an ASCII file.

FDB$OPEN

This procedure provides a method for opening an FDB file for subsequent access by IGORE applications. The format of this command is:

Calling Sequence

FDB$OPEN, lun [, file_name [, file_status [, read_only]]]

Arguments

LUN
An integer specifying the logical unit number to which the FDB file is to be attached.

FILE_NAME
An optional character string parameter specifying the name of the FDB file to be opened. The VMS wildcard file specification is supported, in which case the application produces a list of files from which the user can select a single file. If not provided, the filename will be prompted for.

FILE_STATUS
An optional IDL character string variable which specifies one of following FORTRAN style file status values: "OLD", "NEW", "UNKNOWN", or "SCRATCH". The default is "OLD".

READ_ONLY
An optional IDL variable which specifies readonly access when set to a logical 'true'. If specified and not a logical 'true', the file is opened for read and write access. The default value is readonly.

Discussion

This routine provides a method for opening an FDB file for subsequent access by IGORE applications.

If the logical unit specified by LUN is already attached to a file, an error is issued and the application aborted.

FDB$READ

This procedure provides a method of reading selected data from an FDB file into an IGORE FDR. The format of this command is:

Calling Sequence

FDB$READ, lun, r_fdr [, group [, nrec]]

Arguments

LUN

An integer specifying the logical unit number from which the FDB data are to be read.

R_FDR

An output-only IGORE FDR array into which the data are to be stored.

GROUP

An optional integer specifying the FDB group "tagword" to be read. Currently supported values are given in Table 3.1. The default value of GROUP is 0.

NREC

An optional integer specifying the total number of FDB records to be read. If NREC ≤ 0, the complete FDB group is read. Otherwise, if NREC > 0, only NREC records are read. Note that if NREC is greater than the number of records in the initial group, records from the next group are read until NREC records have been read. The default value of NREC is 0.

Value	First record to be read
< 0	First record in first group in FDB file
$= 0$	First record in next group in FDB file
> 0	First record in specified group "tagword".

Table 3.1: *GROUP* Keyword Values

Discussion

The FDB file must be opened using FDB$OPEN prior to using this application. If R_FDR exists prior to calling this application, any data stored in R_FDR are overwritten. Note that FDB$READ will generally read the FDR's in the same order in which they were written by FDB$WRITE, but this is not guaranteed.

FDB$READALL

This function provides a method of reading *all* data from an FDB file into an IGORE FDR. The format of this command is:

Calling Sequence

nfdr = FDB$READALL(lun, r_fdr)

Arguments

LUN
An integer specifying the logical unit number from which the data are to be read.

R_FDR
An output-only IGORE FDR array into which the data are to be stored.

Discussion

The function return value indicates the number of FDR's read. The FDB file must be opened using FDB$OPEN prior to using this application. If R_FDR exists prior to calling this application, any data stored in R_FDR are overwritten. The FDB tagword cannot be specified; all of the FDB records are read and placed into R_FDR. Note that FDB$READALL will generally read the FDR's in the same order in which they were written by FDB$WRITE, but this is not guaranteed.

FDB$WRITE

This procedure provides the ability to write the information stored in an IGORE FDR to a selected FDB file. The format of this command is:

Calling Sequence

FDB$WRITE, lun, r_fdr

Arguments

LUN
An integer specifying the logical unit number to which the data are to be written.

R_FDR
An IGORE FDR pointer array or subarray identifying the records to be written to the FDB file.

Discussion

The FDB file must be opened using FDB$OPEN with the READ_ONLY parameter set to a logical 'false' prior to using FDB$WRITE.

The FDB tagwords are automatically calculated by this routine. This routine may also be used to add records to the existing file. Note that FDB$READ will generally read the FDR in the same order in which they were written by FDB$WRITE, but this is not guaranteed.

FDR$CREATE

This procedure creates an IGORE FDR array having the specified name and number of elements. The format of this command is:

Calling Sequence

FDR$CREATE, r_fdr, ndims

Arguments

R_FDR

The IDL variable into which the IGORE FDR pointers will be placed. The IGORE record created by this routine will be of type IGR$FDR.

NDIMS

An integer specifying the number of elements in the record array to be created.

Keywords

NAME

A character string which, if present, specifies the name of the IGORE record to be created. Will be truncated, if longer, to 15 characters. If not provided, the IGORE record will be created using the name of the IDL variable passed as R_FDR. This name also will be truncated, if longer, to the first 15 characters.

/UNIQUE

If present and non-zero, /UNIQUE forces the generation of a unique name for the IGORE record. This keyword is ignored if NAME is specified.

Discussion

This routine creates an IGORE record array having a structure type IGR$FDR. Note that FDR$CREATE performs the same operation as the routine CREREC with the structure type IGR$FDR. An example of the usage of FDR$CREATE is given below:

```
Igore> FDR$CREATE, r_fdr1, 10, name='fdr1'
Igore> FDR$CREATE, r_fdr2, 10
Igore> CREREC, 'IGR$FDR', r_fdr3, 10, name='fdr3'
Igore> CREREC, 'IGR$FDR', r_fdr4, 10
```

NOTE: IGORE does not currently support record arrays of more than one dimension.

FDR$FROM_NML

This procedure provides a method of accessing the NAMELIST facility to initialize or modify an FDR fit definition. The format of this command is:

Calling Sequence

FDR$FROM_NML, r_fdr

Arguments

R_FDR
An IGORE FDR pointer array or subarray which is to be initialized or modified.

Discussion

This routine can be used to restore fit definitions from previously saved NAMELIST files (*e.g.* from an INTERACTIVE_GRAPHICS session) and/or interactively modify the fit definition and parameters. Note that not all of the information in an FDR is stored in the NAMELIST file (*e.g.* the line model), however the required information will be prompted for if the FDR is used by other IGORE applications.

FDR$TO_NML

This procedure provides a method of storing an IGORE FDR fit definition in a NAMELIST file. The format of this command is:

Calling Sequence

FDR$TO_NML, r_fdr

Arguments

R_FDR
An IGORE FDR pointer array or subarray having a fit definition which is to be stored in a NAMELIST file.

Discussion

This routine uses the standard NAMELIST facilities to store the fit information for each of the elements in the passed FDR array. The fit definition and parameters can be interactively modified prior to storing the fit information. Note that not all of the information in an FDR is stored in the NAMELIST file (*e.g.* the line model), however the required information will be prompted for if the NAMELIST file is used by other IGORE applications.

FFOLD_SUM

This procedure displays a summary of fit results from data in a model definition record.

Calling Sequence

FFOLD_SUM, mdldef, [ilun]

Arguments

MDLDEF
Model definition record containing best-fit results, as after a call to DO_FOLDANAL.

ILUN
Optional specification of output logical unit number to which the summary results should be written. If not present, results are written to the screen.

Keywords

OUTTEXT
Name of variable to receive output string array containing the fit results.

ABS_UNC
Input switch which if present and nonzero, specifies that the actual duper'd Pmin and Pmax should be displayed instead of the difference in Pmin and Pmax from the best fit.

TARGNAME
Input string identifying the target by name. By default no name is included.

ARRAY_NDX
Input index or indices corresponding to the MDLDEF elements to be displayed. It allows model elements to be identifed by specific numbers in the display. By default model elements are identified by array index.

Discussion

This procedure displays a summary of fit results from information contained in input model definition record MDLDEF.

Example

```
Igore> ffold_sum,ms

Model Element:  1  Domain:   300
   ChiSqr/Dof:  0.891  NDof =     23     (Prob =  0.611842)
   En Range:    0.052 -  1.003 MeV
   Powerlaw Function
No       P            dP         Pmin-P        Pmax-P   Cnstr  Par_Name
 1   1.3502e-02   3.0143e-04  -4.8354e-04   4.8248e-04  SIGN+  Amplitude
 2  -1.6976e+00   4.9670e-02  -8.3832e-02   8.0911e-02  SIGN-  Pwrlaw Index
 3   1.0000e-01                                         FIXED  Normalizing Energ
y
Igore>
```

FFPREP_SPEC

This procedure ensures monotonically increasing spectral energy channels in the passed SDR, a prerequisite for forward folding analysis.

Calling Sequence

FFPREP_SPEC, sdr

Arguments

SDR

Input/output SDR pointer array or subarray containing input spectra to be processed. Output spectra are returned in SDR, overwriting previous spectra.

Keywords

OUTSDB

String specifying the name of the output SDB file. OUTSDB will create the SDB file and populate it with the corrected OSSE spectral data.

ERSTAT

Output array (1 element per SDR) indicating failure (non-zero) or success (zero) in processing of individual SDRs.

GRATIO

Scalar value for the gain ratio specification. A gain ratio of 2 corresponds to a "times 2" gain (approx. 3 keV/chan in low range). If set, energy channel edges are built by ENERGY_CAL using the value GR, rather than taken from the SDR or from the SDB file.

LOW

Input switch which, if present and non-zero, will specify that only low range energy data (0.06 - 1.5 MeV) should be preserved in the resulting SDR.

MED

Input switch which, if present and non-zero, will specify that only medium range energy data (0.06 - 10 MeV) should be preserved in the resulting SDR.

HIGH

Input switch which, if present and non-zero, will specify that only high range data should be included in output SDR.

NOALIGN

Input switch which, if present and non-zero, will specify that data channels should be selected by energy range rather than channel edges.

FULL
Input switch which, if present and non-zero, will specify that high range data should be included along with low and medium range data. Default includes only low and medium range data.

NOOVERLAP
Input switch which, if present and non-zero, will specify that overlap between low and medium ranges should be removed from the MEDIUM range. By default the overlap in included.

ERANGE
Input two-element vector specfying energy range (in MeV) to be included in resulting SDR. If only a lower energy bound is specified, then upper bound defaults to the maximum energy edge value in the SDR. The resulting SDR will contain data only within the specified energy range.

If this keyword is not set, the default energy range of 0.06 MeV to 10 MeV (or to the highest energy bin available in the medium range) is used.

LOFFSET
Specifies the lowest channel to include from the low range.

MOFFSET
Specifies the lowest channel to include from the medium range.

HOFFSET
Specifies the lowest channel to include from the high range.

LTOPCHAN
Specifies the highest channel to include from the low range.

MTOPCHAN
Specifies the highest channel to include from the medium range.

VERBOSE
Input switch which, if present and non-zero, will specifiy that informational messages should be printed.

Discussion

A single spectral record normally contains, for a single detector, data in three energy ranges (0.05-1.5 MeV or Low; 0.05-10.0 MeV or Medium; and 0.05-200 MeV or High) concatenated into a single array. This procedure deletes the zero channels between the Low and Medium range data, as well as the repetitive Medium range data below 1.5 MeV. This ensures monotonically increasing spectral data with respect to energy. Optionally, high range data can also be included.

This procedure is not required for data in any pulsar mode.

Related Routines

See the related routines DTW_CORRECT, BUILD_RMTRX, and DO_FOLDANAL.

FILE_INFO

This application provides a description of any files opened by one of the IGORE OPEN routines. The format of this command is:

Calling Sequence

FILE_INFO [, lun]

Arguments

LUN
The logical unit for which file information is requested.

Discussion

If LUN is not provided, information about all files opened by the user is provided.

NOTE: This routine will not provide information about files opened by any of the IDL `OPEN` routines. Similarly, the IDL command `HELP/FILES` will not provide information about any of the files opened by any of the IGORE OPEN routines.

FIND_SOURCE

This procedure provides a method for searching the target catalog for entries which match a specified target name. The format of this command is:

Calling Sequence

targ_struc = FIND_SOURCE(targ_name, cat)

[This function is written in IDL]

Arguments

TARG_NAME

An input character string specifying the name of the target to be found. TARG_NAME can be abbreviated, in which case all targets from the catalog CAT satisfying the abbreviation are listed and the user is prompted for the target to be selected.

CAT

An input structure array of type PLANCAT containing the target catalog to be searched.

Keywords

/NOPROMPT

If present and nonzero, /NOPROMPT prevents request for resolution of multiple targets meeting input TARG_NAME. Multiple possibilities cause error message and return of 0. Default action is to prompt for resolution of multiple targets.

ERSTAT

Completion status, non-zero if error(s) occured. Output variable which if present, will report failures rather than check type of return.

Discussion

This function provides a method for searching a source catalog specified by CAT for a selected target and returning information on the selected target in the variable TARG_STRUC. The returned variable TARG_STRUC is a structure of type PLANCAT and contains various information about the selected target.

See the routine READ_PLANCAT for reading the OSSE Target Database to generate a variable of structure type PLANCAT for the argument CAT.

FIND_SOURCE_ID

This function provides a method for searching the target catalog for entries which match a specified target name. The format of this command is:

Calling Sequence

targ_id = FIND_SOURCE_ID(targ_name, cat)

[This function is written in IDL]

Arguments

TARG_NAME

An input character string specifying the name of the target to be found. TARG_NAME can be abbreviated, in which case all targets from the catalog CAT satisfying the abbreviation are listed and the user is prompted for the target to be selected.

CAT

An input structure array of type PLANCAT containing the target catalog to be searched.

Discussion

This function provides a method for searching a source catalog specified by CAT for a selected target and returning the target catalog source ID in the variable TARG_ID.

See the routine READ_PLANCAT for reading the OSSE Target Database to generate a variable of structure type PLANCAT for the argument CAT.

FIND_TARGET_MDP

This function provides a method for searching the command state database over the specified time interval and identifying all command states involving observations of the specified target. The format of this command is:

Calling Sequence

status = FIND_TARGET_MDP(startdate, stopdate, targ_name, state, max_off)

[This function is written in IDL]

Arguments

STARTDATE

A character string specifying the start time of the interval over which the processing is to be performed. The form of STARTDATE is "yy/ddd hh:mm:ss". It may be abbreviated as long as the "yy/ddd" is specified; any missing elements are assumed to be zero.

STOPDATE

A character string specifying the stop time of the interval over which the processing is to be performed. The form of STOPDATE is "yy/ddd hh:mm:ss". It may be abbreviated as long as the "yy/ddd" is specified; any missing elements are assumed to be zero.

TARG_NAME

An input character string specifying the name of the target of interest.

STATE

An output structure array of type STATE_4 containing a description of each of the command states corresponding to observations of the specified target.

MAX_OFF

An input parameter specifying the maximum scan angle, in degrees, for the observation to be considered a "source" observation of the specified target.

Keywords

CAT

An input structure array of type PLANCAT containing the target catalog to be searched. If not provided, the complete OSSE Target Database is used.

MAX_XOFF

An input specifying the maximum xscan angle, in degrees, for the observation to be considered a "source" observation of the specified target. The default value of MAX_XOFF is 10 degrees.

MDP

An output structure array of type MDP_STRUC containing descriptions of the MDP strategy tables for observations of the specified target. There is an element in the array for each returned element of STATE.

TABLETYPE

An input string array of one or more positioning table type names. Valid names are "PRI", "SEC", "SOL" and "TRANS". If one or more positioning table types are specified, only those types specified will be examined. Default (without the keyword) is to examine all four types.

Discussion

This function provides a method for searching the command state database over the specified time interval and identifying all command states which contain observations of the specified target. The returned variable STATUS indicates whether any valid command states were found; a value of STATUS = 1 indicates that one or more command states were found containing observations of the specified target, a value of STATUS = 0 indicates that no observations of the specified target were found within the time interval specified by STARTDATE and STOPDATE and within the scan angle range specified by MAX_OFF.

FLDNAME

This function returns a string array specifying all of the field names for the structure type of the passed IGORE record. The format of this command is:

Calling Sequence

name_array = FLDNAME(r_pointer)

Arguments

R_POINTER
An IGORE record pointer, or single element of a pointer array, for which the field names are desired.

Discussion

This routine returns a string array containing the names of each of the fields in the IGORE record structure corresponding to the structure type of R_POINTER.

Related Routines

See the related routine SHOWFLD, which produces a listing of the field names for a specified IGORE structure type.

FREEVM

This application frees the virtual memory controlled by the IGORE temporary virtual memory management table. The format of this command is:

Calling Sequence

FREEVM

Discussion

This routine is useful if an application aborts without freeing the temporary memory which had been allocated. The memory allocation for the temporary IGORE table can be determined using the command: SHOWIGR, table = 'temporary'.

GETAUX

This function provides the ability to extract a field from an AUX record of an SDR or SDR array. The format of this command is:

Calling Sequence

return = GETAUX(auxkey, field_name, r_sdr [, index])

[This procedure is written in IDL]

Arguments

AUXKEY

An integer value specifying the AUX type from which the field is to be extracted. An AUX record of this type must be present in the SDR.

See the "Spectral Data Base Header Definition" document for a list of the supported AUX key values and the corresponding structure types.

FIELD_NAME

A character string containing the name of the field in the AUX record which is to be extracted. This argument is ignored if the AUX record is an indirect field.

R_SDR

An SDR pointer array or subarray containing the AUX record from which the field is to be extracted.

INDEX

An optional integer parameter specifying the index element of the AUX record array to be extracted. The AUX field FIELD_NAME is returned only for the element of the AUX record array specified by INDEX. This argument is only valid if the specified AUX type is a record array.

Keywords

FILL

A byte variable specifying the value which is used to 'fill' any empty elements of the returned variable. The default value of FILL is 0. This keyword is only valid when DIMSIZE = 1, which is the default.

DIMSIZE

An integer specifying the method for determining the dimensionality of the output IDL variable RETURN. The default value of DIMSIZE is 1. The supported values of DIMSIZE are:

0 – The output variable RETURN is dimensioned (AUXD1 [, AUXD2, ...], NPTR), where NPTR represents the number of elements in the SDR array R_SDR and AUXD1 [, AUXD2, ...] represents the dimensionality of the specified AUX field as determined from the **first** element of R_SDR.

1 – The output variable RETURN is dimensioned (AUXD1 [, AUXD2, ...], NPTR), where NPTR represents the number of elements in the SDR array R_SDR and AUXD1 [, AUXD2, ...] represents the dimensionality of the specified AUX field as determined from the SDR containing the **largest** dimensioned AUX data field. For AUX data fields smaller than the maximum size, the empty elements are 'filled' using the value specified by the FILL keyword. The default 'fill' value is 0.

−1 – The output variable RETURN is dimensioned (NTOTAUX), where NTOTAUX represents **the total number of AUX data elements contained in the input SDR array** R_SDR.

NOTE: The correct specification of DIMSIZE **is important when both the** AUX **field and** R_SDR **are arrays. See the example below.**

Discussion

For the passed SDR array R_SDR, this function extracts the information stored in the field FIELD_NAME of the AUX record specified by AUXKEY and returns the result in the IDL variable RETURN. If the specified AUX type is not present in **all** of the passed SDRs no error is reported. For each of the SDR elements which do not contain the specified AUX type, fill records are used (default to byte value 0.) If the specified AUX type is not present in **any** of the passed SDRs, an error is reported and the routine is aborted.

An example of the usage of GETAUX is given below:

```
Igore> SDR$CREATE, r_sdr, 2
Igore> CREREC, 'IGR$AU_DEF', r_audef, 2
Igore> CREREC, 'IGR$AUID', r_auid, 5
Igore> PUTFLD, 'SRC_DT', r_audef(0), [0.0, 1.0, 2.0, 3.0]
Igore> PUTFLD, 'SRC_DT', r_audef(1), [4.0, 5.0, 6.0, 7.0]
Igore> PUTFLD, 'TAG', r_auid, [1, 2, 3, 4, 5]
Igore> PUTAUXREC, !AUX.AUDEF, r_sdr(0), auxrec=r_audef(0)
Igore> PUTAUXREC, !AUX.AUDEF, r_sdr(1), auxrec=r_audef(1)
Igore> PUTAUXREC, !AUX.AUID, r_sdr(0), auxrec=r_auid(0:3)
Igore> PUTAUXREC, !AUX.AUID, r_sdr(1), auxrec=r_auid
Igore> PRINT, GETAUX( !AUX.AUDEF, 'SRC_DT', r_sdr(0), dimsize=0)
      0.000000      1.00000      2.00000      3.00000
Igore> PRINT, GETAUX( !AUX.AUDEF, 'SRC_DT', r_sdr(1), dimsize=0)
       4.00000      5.00000      6.00000      7.00000
Igore> PRINT, GETAUX( !AUX.AUDEF, 'SRC_DT', r_sdr, dimsize=0)
      0.000000      1.00000      2.00000      3.00000
       4.00000      5.00000      6.00000      7.00000
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr(0), dimsize=0)
       1           2           3           4
```

```
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr(1), dimsize=0)
         1           2           3           4           5
Igore> Print, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=0)
         1           2           3           4
         1           2           3           4
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr)
         1           2           3           4           0
         1           2           3           4           5
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, fill=-1)
         1           2           3           4          -1
         1           2           3           4           5
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=-1)
          1           2           3           4           1           2
          3           4           5
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, 1, dimsize=0)
          1           1
```

In the above example, three records are created; R_SDR, R_AUDEF and R_AUID. The fields SRC_DT and TAG are filled in the records R_AUDEF and R_AUID, respectively. These records are then placed into the AUX fields of R_SDR with AUXKEYs specified by `!AUX.AUDEF` and `!AUX.AUID`, respectively. Note that elements R_AUID (0:3) are placed into element 0 of R_SDR while elements R_AUID (0:4) are placed into element 1 of R_SDR. Next, GETAUX is used to extract the SRC_DT field from the `!AUX.AUDEF` AUX record of R_SDR. Next, GETAUX is used to extract the TAG field from the `!AUX.AUID` AUX record of R_SDR. Note that four elements are returned from R_SDR(0) while five elements are returned from R_SDR(1). When GETAUX is called with both elements of R_SDR, the returned array is dimensioned (4,2) since TAG has only four elements in R_SDR(0). When the same GETAUX call is made with DIMSIZE=1, the returned array is dimensioned (5,2) since the maximum dimension of TAG is five (contained in R_SDR(1)). Note that, in this example, element (5,1) of the returned array is 'filled' with 0. The next command shows the use of the FILL keyword where now element (5,1) of the returned array is 'filled' with the value −1, as specified in the command line. Calling GETAUX with DIMSIZE=−1 returns an array dimensioned (9), corresponding to the total number of elements in TAG stored in R_SDR. Finally, the optional argument INDEX is used to extract only element 1 of the TAG arrays from each element of R_SDR.

Note that the function GETAUX can only be used to extract a **single field** from an AUX record of an SDR array; the **entire AUX record** can be extracted from an SDR element using the function GETAUXREC.

Related Routines

Also see the related function GETAUXREC.

GETAUXREC

This function provides the ability to extract an entire AUX record from an SDR. The format of this command is:

Calling Sequence

return_pointers = GETAUXREC(auxkey, r_sdr [, index])

[This procedure is written in IDL]

Arguments

AUXKEY

An integer value specifying the AUX record type which is to be extracted. An AUX record of this type must be present in the SDR.

See the "Spectral Data Base Header Definition" document for a list of the supported AUX key values and the corresponding structure types.

R_SDR

An SDR array or subarray containing the AUX record(s) to be extracted.

INDEX

An optional integer parameter specifying the index element of the AUX record array to be extracted. The AUX record is returned only for the element of the AUX record array specified by INDEX. This keyword is only valid if the specified AUX type is a record array.

Keywords

NAME

A character string specifying the name to be used for the output IGORE record. If this keyword is not provided, a unique IGORE record name will be generated for the output record.

FILL

A byte variable specifying the value which is used to 'fill' any empty elements of the returned record. The default value of FILL is 0. This keyword is only valid when DIMSIZE = 1, which is the default.

/HEADER

If present and non-zero, /HEADER indicates that, for a "Type 3" AUX record, the returned pointers RETURN_POINTERS will point to the "header" portion of the specified records. The default is for the returned pointers RETURN_POINTERS to point to the "data" portion of the AUX record. This keyword is only valid if the

AUX record is a "Type 3" record consisting of a "header" portion and a "data" portion.

DIMSIZE
An integer specifying the method for determining the dimensionality of the output IDL pointer variable RETURN_POINTERS. The default value of DIMSIZE is 1. The supported values of DIMSIZE are:

0 – The output pointer RETURN_POINTERS is dimensioned (AUXD1 [, AUXD2, ...], NPTR), where NPTR represents the number of elements in the SDR array R_SDR and AUXD1 [, AUXD2, ...] represents the dimensionality of the specified AUX records it as determined from the **first** element of R_SDR.

1 – The output variable RETURN_POINTERS is dimensioned (AUXD1 [, AUXD2, ...], NPTR), where NPTR represents the number of elements in the SDR array R_SDR and AUXD1 [, AUXD2, ...] represents the dimensionality of the specified AUX record as determined from the SDR containing the **largest** dimensioned AUX record. For AUX records smaller than the maximum size, all fields of the empty elements are 'filled' using the value specified by the FILL keyword. The default 'fill' value is 0.

−1 – The output variable RETURN_POINTERS is dimensioned (NTOTAUX), where NTOTAUX represents **the total number of AUX records contained in the input SDR array** R_SDR.

NOTE: **The correct specification of** DIMSIZE **is important when both the** AUX **record and** R_SDR **are arrays. See the example below.**

Discussion

For the passed SDR R_SDR, this function extracts the AUX records specified by AUXKEY and returns the pointers to the records in the IDL variable RETURN_POINTERS. If the specified AUX type is not present in **all** of the passed SDRs no error is reported. For each of the SDR elements which do not contain the specified AUX type, fill records are used (default to byte value 0.) If the specified AUX type is not present in **any** of the passed SDRs, an error is reported and the routine is aborted.

Continuing the GETAUX example above, an example of the usage of GETAUXREC is given below:

```
Igore> tst0 = GETAUXREC( !AUX.AUID, r_sdr(0), name='r_tst0', dimsize=0)
Igore> tst1 = GETAUXREC( !AUX.AUID, r_sdr(1), name='r_tst1', dimsize=0)
Igore> tst2 = GETAUXREC( !AUX.AUID, r_sdr, name='r_tst2', dimsize=0)
Igore> tst3 = GETAUXREC( !AUX.AUID, r_sdr, name='r_tst3')
Igore> tst4 = GETAUXREC( !AUX.AUID, r_sdr, name='r_tst4', fill=-1)
Igore> tst5 = GETAUXREC( !AUX.AUID, r_sdr, name='r_tst5', dimsize=-1)
Igore> tst6 = GETAUXREC( !AUX.AUID, r_sdr, 1, name='r_tst6', dimsize=0)
Igore> HELP, tst0, tst1, tst2, tst3, tst4, tst5, tst6
TST0            STRUCT    = -> IGR$REC_PTRS Array(4)
```

```
TST1            STRUCT    = -> IGR$REC_PTRS Array(5)
TST2            STRUCT    = -> IGR$REC_PTRS Array(4, 2)
TST3            STRUCT    = -> IGR$REC_PTRS Array(5, 2)
TST4            STRUCT    = -> IGR$REC_PTRS Array(5, 2)
TST5            STRUCT    = -> IGR$REC_PTRS Array(9)
TST6            STRUCT    = -> IGR$REC_PTRS Array(2)
Igore> showrec
Name            Dims  Structure            RecL        Direct      Indirect
R_AUDEF            2  IGR$AU_DEF            453           906             0
R_AUID             5  IGR$AUID               29           145             0
R_SDR              2  IGR$SDR               150           300          1331
R_TST0             4  IGR$AUID               29           116             0
R_TST1             5  IGR$AUID               29           145             0
R_TST2             8  IGR$AUID               29           232             0
R_TST3            10  IGR$AUID               29           290             0
R_TST4            10  IGR$AUID               29           290             0
R_TST5             9  IGR$AUID               29           261             0
R_TST6             2  IGR$AUID               29            58             0
Igore> PRINT, GETFLD( 'TAG', tst0)
           1           2           3           4
Igore> PRINT, GETFLD( 'TAG', tst1)
           1           2           3           4           5
Igore> PRINT, GETFLD( 'TAG', tst2)
           1           2           3           4
           1           2           3           4
Igore> PRINT, GETFLD( 'TAG', tst3)
           1           2           3           4           0
           1           2           3           4           5
Igore> PRINT, GETFLD( 'TAG', tst4)
           1           2           3           4          -1
           1           2           3           4           5
Igore> PRINT, GETFLD( 'TAG', tst5)
           1           2           3           4           1           2
           3           4           5
Igore> PRINT, GETFLD( 'TAG', tst6)
           1           1
```

In the above example, the AUX records corresponding to AUXKEY = `!AUX.AUID` are extracted from the SDR array R_SDR as it was generated in the GETAUX example. First, the AUX records from R_SDR(0) and R_SDR(1) are extracted separately, with the output record names R_TST0 and R_TST1, respectively. The IDL variables containing these record pointers are TST0 and TST1. Note that R_TST0 contains four elements and R_TST1 contains five elements. When GETAUXREC is called with both elements of R_SDR, the returned pointer array TST2 is dimensioned (4,2) since the AUX record array has only four elements in R_SDR(0). When the same GETAUXREC call is made with DIMSIZE=1, the returned pointer array TST3 is dimensioned (5,2) since the maximum dimension of the AUX array is five (contained in R_SDR(1)). Note that, in this example, field (5,1) of TST3 is filled with 0, as shown by the GETFLD('TAG', tst3) command. The next command shows the use of the FILL keyword, where now field (5,1) of TST3 is filled with −1, as shown by the GETFLD('TAG', tst4) command. Calling GETAUXREC with DIMSIZE=−1 returns an array dimensioned (9), corresponding to the total number of AUX array elements stored in

R_SDR. Finally, the optional argument INDEX is used to extract only element 1 of the AUX record arrays from each element of R_SDR.

NOTE: While the pointer array RETURN_POINTERS returned by GETAUXREC may have more than one dimension, the actual IGORE records are stored as one-dimensional arrays. If the command POINTERS is used to restore such a record, or if such a record was saved using SAVE_REC and subsequently restored using RESTORE_REC, the resulting IDL pointer array would be restored as a one-dimensional array.

Related Routines

Also see the related function GETAUX.

GETFLD

This function returns data stored in an IGORE record or record array into a local IDL variable. The format of this command is:

Calling Sequence

return = GETFLD(field_name, r_pointers)

Arguments

FIELD_NAME

A character string specifying the IGORE structure field name of the data to be extracted.

R_POINTERS

An IGORE record array or subarray for which the specified field data are to be extracted.

Keywords

Same as SDR$GETFLD.

Discussion

For the passed record pointer R_POINTERS, this routine returns the information stored in the IGORE structure field specified by FIELD_NAME into the local IDL variable RETURN.

If the field value specified by FIELD_NAME is a scalar, the return value will be dimensioned (NPTR1 [,NPTR2,...]), where NPTR1 [,NPTR2,...] represents the dimensionality of the pointer array R_POINTERS.

If the field value specified by FIELD_NAME is an array, but not an indirect field, the return value array will be dimensioned (DIM1 [,DIM2,...], NPTR1 [,NPTR2,...]), where NPTR1 [,NPTR2,...] represents the dimensionality of the pointer array R_POINTERS and DIM1 [,DIM2,...] represents the dimensionality of the specified field.

If FIELD_NAME represents an indirect field, the return value will be dimensioned (DIM1 [,DIM2,...], NPTR1 [,NPTR2,...]), where NPTR1 [,NPTR2,...] represents the dimensionality of the pointer array R_POINTERS and DIM1 [,DIM2,...] represents the dimensionality of the specified field **as determined from the first element of** R_POINTERS.

Related Routines

See the related routines PUTFLD and SDR$GETFLD.

GETIFLD

This function returns the data stored in the indirect access field of an IGORE record. The format of this command is:

Calling Sequence

variable = GETIFLD(field_pointer)

Arguments

FIELD_POINTER
An IGORE indirect field record pointer, or single element of a pointer array, for which the indirect field data are to be extracted.

Discussion

This routine can be used to return the data stored in an indirect field of an IGORE record. The field pointer can be obtained using the "@FIELD_NAME" feature in the command GETFLD. The following example shows the use of this option of GETFLD to get field pointers, along with the use of those pointers by GETIFLD to get the data.

```
Igore> DAT_FIELD_PTRS=GETFLD('@DAT',SDR)
Igore> DATA=GETIFLD(DAT_FIELD_PTRS)
Igore> DATA2=GETFLD('DAT',SDR)
```

In the above example, DATA and DATA2 will contain the same information.

IGORE_DATE

This procedure prints the creation date of the active IGORE shareable image. The format of this command is:

Calling Sequence

IGORE_DATE

Discussion

This routine is useful in verifying that user applications have been linked using the most recent version of the IGORE shareable image.

IGR$CLOSE

This procedure closes any file previously opened by any of the IGORE OPEN routines. The format of this command is:

Calling Sequence

IGR$CLOSE, lun

Arguments

LUN
An integer parameter specifying the logical unit number to close.

Discussion

This routine closes the logical unit specified by LUN. This routine also frees the appropriate logical units from the logical unit table, where necessary. Note that this routine will **not** close files opened by IDL.

This is the preferred method for closing IGORE files.

IGR$FREE_LUN

This function frees a FORTRAN unit allocated by IGR$GET_LUN or SDB$OPEN.

The format of this command is:

Calling Sequence

return = IGR$GET_LUN(lun)

Arguments

LUN
Variable containing FORTRAN logical unit number to be freed.

Keywords

None

Discussion

This function frees a FORTRAN unit allocated by IGR$GET_LUN or SDB$OPEN, returned the completion status. Unlike the VMS library routine LIB$FREE_LUN, IGR$FREE_LUN will free an allocation recorded in the table used by the IGORE file access routines.

Note: If a file is open on LUN, the file is closed before LUN is freed.

Related Routines

See the related routines IGR$GET_LUN, and SDB$OPEN.

IGR$REC_COMPARE

This function compares IGORE record fields stored in two records, displaying a list of the differences. The format of this command is:

Calling Sequence

return = IGR$REC_COMPARE(rec1, rec2, struc)

[This procedure is written in IDL]

Arguments

REC1
An input IGORE record pointer, or pointer array element, for which the record comparison is to be performed. Assumes name is no longer than 15 characters.

REC2
An input IGORE record pointer, or pointer array element, for which the record comparison is to be performed. Assumes name is no longer than 15 characters.

STRUC
An input character string containing the IGORE structure name.

Keywords

DIFFS
A output character string array containing the names of the mismatching fields.

EXCLUDE
An input character string array containing the names of fields to exclude from the comparison.

/VERBOSE
If set, displays a list of mismatching fields.

Discussion

This function returns in RETURN the result of the compare. RETURN equals 1 if the records match and equals 0 if there are differences

IGR$REC_COMPARE looks up the IGORE record structure specified by STRUC, extracts the list of field names, excludes the field names designated by EXCLUDE, extract the data field by field from REC1 and REC2 and compares them. A list of mismatching fields is either displayed based on the /VERBOSE keyword and/or returned in the keyword character string array, DIFFS. If no mismatches are detected, DIFFS returns a 1 elements string array containing a null.

Example

```
Igore> stat1 = igr$rec_compare,rec1,rec2,'IGR$SDB_HDR',/verbose,diffs=diffs,$
Igore> exclude='CREDATE'

Comparing records with structure: igr$sdb_hdr
        Record             1                      2
  #      Field     --------------------  --------------------

  4     TAGWORD                     101                   102
 17   DETCFG_ID                    1866                  2137

IGORE>print,diffs
TAGWORD DETCFG_ID
Igore>
```

Related Routines

See the related routine SDR_HDR_COMPARE.

IGRHLP

This procedure provides a method for interactively accessing the IGORE HELP facility. The format of this command is:

Calling Sequence

IGRHLP [, topic]

Arguments

TOPIC
An optional character string specifying the name of the IGORE command for which HELP is requested.

Discussion

This routine provides an interactive user interface into the IGORE HELP facility, providing a description of the available IGORE commands including their calling format, parameters and keywords. If the optional parameter TOPIC is provided, HELP for the specified command is listed; otherwise, the general HELP facility is activated and the user is prompted for the name of the command for which HELP is required. This HELP facility uses the VMS HELP Facility.

For help on IDL system commands, use the IDL on-line help command by typing `?` at the `Igore>` prompt. For help on functions and procedures written in IDL, use the IDL library documentation facilities `DOC_LIBRARY` or `XDL`.

IMSG

This procedure formats and displays a message based on input parameters.

Calling Sequence

IMSG, severity, text

[This procedure is written in IDL]

Arguments

SEVERITY
An input integer specifying the severity of the message to be issued. "0" specifies a fatal error, "1" specifies a warning message, and "2" specifies an informational message.

TEXT
An input character string containing the message text.

Keywords

CALLER
An input character string to specify the calling routine name.

LUN
An input integer to specify message output lun.

/VERBOSE
An input logical switch if present and nonzero specifies that informational message should be printed.

/TRACEBACK
An input logical switch if present and nonzero specifies that the traceback should be printed.

LCDISPLAY

This procedure displayes a one page summary of the data in an EFOLD record. The format of this command is:

Calling Sequence

LCDISPLAY, r_efold [, output_file]

[This procedure is written in IDL]

Arguments

R_EFOLD
An EFOLD pointer, or single element of an array, containing the pulsar light curve to display.

OUTPUT_FILE
An optional character string specifying the file to which the summary will be written. If no output file is specified, information is written to the screen.

Discussion

After EFOLD is run on a record, this command will display information about the pulsar data and light curve, including the pulsar window set up, the pulsar mode, time resolution, start and stop time of the accumulated data, etc.

Detector pointing information can be displayed using LCPOINT.

LCPOINT

This procedure displays a summary of the pointings in an EFOLD record. The format of this command is:

Calling Sequence

LCPOINT, r_efold [, output_file]

[This procedure is written in IDL]

Arguments

R_EFOLD
An EFOLD pointer, or single element of an array, containing the pulsar light curve to display.

OUTPUT_FILE
An optional character string specifying the file to which the summary will be written. If no output file is specified, information is written to the screen.

Discussion

After EFOLD is run on a record, this command will display information about the detector pointings during the accumulation interval, including the separation angle between the pulsar and the center of the collimator.

Information about the pulsar mode and window set up can be displayed using LCDISPLAY.

LCSUM

This procedure extracts a light curve from an EFOLD record, sums the selected data into a single vector, and returns the result. The format of this command is:

Calling Sequence

LCSUM, r_efold, lc

[This procedure is written in IDL]

Arguments

R_EFOLD
An EFOLD pointer, or single element of an array, containing the pulsar light curve to sum.

LC
The output light curve. The parameter LC is an I*4 vector with NPHASBIN elements, where NPHASBIN is defined in the input EFOLD record.

Discussion

The light curve produced by EFOLD is either a 3- or 5-dimensional matrix, depending on the instrument pulsar mode (see the Pulsar Analysis Cookbook). This procedure is used to select a subset of this multi-dimensional array and sum it into a single light curve. Depending on the configuration of the instrument, the user is prompted to select detector number(s), window number(s), energy channels within selected windows, and pointing(s). The user may respond with a list of numbers or the word 'all' to select all values in a specific category. The output light curve is **not** live-time normalized.

LISTREC

This procedure lists the contents of an IGORE record or record array. The format of this command is:

Calling Sequence

LISTREC, r_pointers

[This procedure is written in IDL]

Arguments

R_POINTERS
An IGORE record pointer, or pointer array, for which the listing is to be performed.

Discussion

This routine lists the contents of all the fields of the IGORE record or record array specified by R_POINTERS. Indirect access fields and subfields of indirect access records are also displayed. Uninitialized indirect access records and indirect access fields are specified by the comments "**Empty Record**" and "**Empty Field**", respectively.

LOADSTRUC

This procedure provides the ability to load selected IGORE structures from a text library. The format of this command is:

Calling Sequence

LOADSTRUC, file_name

[This procedure is written in IDL]

Arguments

FILE_NAME

A character string specifying the text library file containing the structures to be loaded.

Keywords

STRUCTURE

A character string specifying the structures to be extracted from the text library. This parameter may specify a single name of a structure (max 15 chars), or represent multiple structures either by specifying a list of names separated by commas or by passing a string array of individual structure names.

/SHOW

If present and non-zero and a single structure has been selected, /SHOW specifies that the definition of the structure is to be displayed. The default action is to produce no display of the structure.

Discussion

This routine will load the structure descriptor(s) specified by STRUCTURE from the text library specified by FILE_NAME for the current IGORE session.

NOTE: User defined structures cannot be accessed by GETAUX, PUTAUX, etc.

MERGECHAN

This procedure provides a method for creating a matrix which maps specified input channels into desired output bins. The format of this command is:

Calling Sequence

MERGECHAN, edg1, wid1, edg2, wid2, matrix, oedg2, owid2

[This procedure is written in IDL]

Arguments

EDG1
An input array of the initial bin lower edges.

WID1
An input array of the initial bin widths.

EDG2
An input array of the desired bin lower edges.

WID2
An input array of the desired bin widths.

OEDG2
An output array of the resultant bin lower edges.

OWID2
An output array of the resultant bin widths.

MATRIX
An output matrix, dimensioned (NEDG2, NEDG1) where NEDG1 is the number of elements in EDG1 and NEDG2 is the number of elements in EDG2, which maps the input bins into the output bins.

Keywords

/NATURAL
If present and non-zero, /NATURAL turns off the warning message that the input edge definition is not monotonic increasing.

OUTMAP
Output array providing mapping of input channels to binned output channels. Array has one entry per input edge and elements contain channel number of output edge.

RANGE_SWITCH
An input 2-element array which defines the energies at which binning changes from LOW to MEDIUM and from MEDIUM to HIGH spectral ranges. The default is to take the highest channel of each range (*e.g.*, use all of the LOW range, then the remaining portion of the MEDIUM range).

Related Routines

Also see the related function RESAMPLE.

MERGEREC

This application copies a non-contiguous IGORE record pointer array into a contiguous record array. The format of this command is:

Calling Sequence

MERGEREC, r_source, r_destination

[This procedure is written in IDL]

Arguments

R_SOURCE

The input IGORE record pointer array or subarray to be copied. The records referenced by this array may be non-contiguous, but must all be of the same structure type. Name is assumed to be no longer than 15 characters.

R_DESTINATION

The returned pointer array into the contiguous IGORE record array into which the input record array was copied. Name is truncated if longer than 15 characters.

Keywords

NEW_NAM

A character string specifying the name for the IGORE output record. Will be truncated if longer than 15 characters.

/UNIQUE

If present and non-zero, /UNIQUE forces the generation of a unique name for the IGORE output record. This keyword is ignored if the NEW_NAM keyword is present.

Discussion

This routine copies a non-contiguous IGORE record pointer array specified by R_SOURCE into a contiguous record array specified by R_DESTINATION. The input record pointers must all reference records of the same structure type. The name of the IGORE output record is taken from the name of the IDL variable passed as R_DESTINATION. This will be truncated if it is longer than 15 characters. Since MERGEREC actually copies the records, not just the pointers, from R_SOURCE to R_DESTINATION, the original records may be deleted without affecting records stored in R_DESTINATION. An example of the usage of MERGEREC is given below:

```
Igore> SDR$CREATE, r_sdr1, 10
Igore> SDR$CREATE, r_sdr2, 10
```

```
Igore> index = [1, 3, 5, 7, 9]
Igore> MERGEREC, [r_sdr1(index), r_sdr2(index)], r_sdr3
Igore> sdr3 = [r_sdr1(index), r_sdr2(index)]
Igore> DELREC, sdr3, /all
```

In the above example two SDR records are created, each having ten elements. The odd elements of the record arrays R_SDR1 and R_SDR2 are merged into a single record array R_SDR3. The pointers for the same odd elements from record arrays R_SDR1 and R_SDR2 are also copied into the IDL variable SDR3. Finally, the /ALL keyword is used with DELREC to delete **all** of the records referenced in SDR3. Note that this command deletes the record arrays R_SDR1 and R_SDR2; however, the original IDL variables containing pointers to these records still exist but no longer point to valid records. The record array R_SDR3 still contains the information from records R_SDR1 and R_SDR2 since MERGEREC copied the selected records (not just the pointers) into R_SDR3.

MODELSTRUC

MODELSTRUC is low-level utility function which returns an unpopulated model definition record.

Calling Sequence

return = MODELSTRUC()

Arguments

None

Keywords

None

Discussion

MODELSTRUC is low-level utility function which returns an unpopulated model definition record. Each model record, an IDL record of the structure MODSTR, defines a single component function of a photon model. An array of model records can define a model composed of multiple functions.

Information stored in the model definition record includes initial values for each function parameter, best fit values for each parameter, and statistics such as parameter uncertainties, χ^2, and NDOF, as well as a name indentifying the IDL routine which implements the function.

Not generally useful to the data analsis user, MODELSTRUC is described here to provide insight into the model structure MODSTRUC and model record usage.

Related Routines

See the related routines BUILDMODEL, MODEL_READ, and MODEL_WRITE.

MODEL_READ

This function reads model data from a namelist file, and returns a populated model definition record (or record array) for use in forward folding analysis.

Calling Sequence

return = MODEL_READ(modelfile)

Arguments

MODELFILE
Name of namelist file containing the model parameters. The default extension for this file is ".MODEL".

Discussion

This functions reads model data from the namelist file specified in MODELFILE and returns a populated model definition record (or record array) suitable for use by the forward folding analysis procedures. The namelist file may be created either with the MODEL_WRITE function or with a text editor.

Related Routines

See the related routines BUILDMODEL, MODEL_WRITE and MODELSTRUC.

MODEL_WRITE

This procedure writes data from a model definition record (or record array), to an ASCII formatted namelist file.

Calling Sequence

return = MODEL_WRITE(modelfile, model)

Arguments

MODELFILE

Output file name for the model. A directory specification may be included in the string.

MODEL

Input model definition record (or record array). The model record(s) are typically created by BUILDMODEL or read by MODEL_READ.

Discussion

This procedure writes data from a model definition record (or record array), to an ASCII formatted namelist file. The resulting file may be edited or supplied as input to the forward folding routines.

Related Routines

See the related routines BUILDMODEL, MODEL_READ and MODELSTRUC.

MOVEIND

This application copies a direct or indirect record from one IGORE record pointer to another. There are several formats for this command:

Calling Sequence

MOVEIND, in_field, r_in_pointer, r_out_pointer
MOVEIND, r_in_pointer, out_field, r_out_pointer
MOVEIND, in_field, r_in_pointer, out_field, r_out_pointer

Arguments

IN_FIELD
A character string containing the name of the indirect record to be extracted from the input IGORE record pointer.

OUT_FIELD
A character string containing the name of the indirect record into which the specified field is to be copied.

R_IN_POINTER
The IGORE record pointer, or single element of a pointer array, from which the input indirect field/record is to be obtained.

R_OUT_POINTER
The IGORE record element, or single element of a pointer array, into which the indirect field/record is to be placed.

Discussion

This routine is used to copy an indirect access record from one IGORE record to another. The type of copy executed depends upon the arguments passed to the routine. The supported modes of copying indirect records are:

- **Copying an indirect record to a direct record.**

  The command format for this copying mode is:

  MOVEIND, in_field, r_in_pointer, r_out_pointer

  In this mode, the indirect access record specified by IN_FIELD is copied from the IGORE record R_IN_POINTER into the direct access record specified by R_OUT_POINTER. The record type of R_OUT_POINTER must be the same as the indirect field IN_FIELD. If the record specified by R_OUT_POINTER does not exist and it is passed with a valid, unused record name, it will be created.

- **Copying a direct record to an indirect record.**

The command format for this copying mode is:

MOVEIND, r_in_pointer, out_field, r_out_pointer

In this mode, the direct access record specified by R_IN_POINTER is copied into the indirect access record OUT_FIELD of the IGORE record R_OUT_POINTER. The record type of the output field in R_OUT_POINTER must be an indirect field of the same type as the record R_IN_POINTER.

- **Copying an indirect record to an indirect record.**

  The command format for this copying mode is:

  MOVEIND, in_field, r_in_pointer, out_field, r_out_pointer

  In this mode, the indirect access record of R_IN_POINTER specified by IN_FIELD is copied into the indirect access record OUT_FIELD of R_OUT_POINTER. The record types of the input and output fields IN_FIELD and OUT_FIELD must be indirect fields of the same type.

MTRX_REBIN

This procedure resamples both the input SDR and its corresponding response matrix into the desired bin sizes.

Calling Sequence

MTRX_REBIN, sdr, edges, widths, outsdb

Arguments

SDR

Input-ouput SDR containing data to be rebinned. In the preferred usage, this SDR, or its parent SDB file, will also contain the required response matrix data. The data is rebinned in place, overwriting the existing data.

EDGES

Float array containing the rebinned energy edges into which the spectrum data and response matrix will be resampled. Ignored if keyword NBINS is present. Single dimensional.

WIDTHS

Float array containing rebinned energy widths corresponding to the rebinned energy edges found in EDGES. Ignored if keyword NBINS is present. Single dimensional.

OUTSDB

The argument OUTSDB specifies the name for an output SDB file to receive the rebinned spectral data and response matrix.

Keywords

ERSTAT

Completion status (0 - success, 1 - failure).

ERANGE

If NBINS is a vector, NBINS must be vector specifying the energy range for each NBINS element in units of MeV. This parameter is ignored if NBINS is a scalar. The number of elements in ERANGE must be one greater than the number of elements in NBINS. For example, to resample spectrum between 0.05-1.5 MeV to 5 bins and spectrum between 1.5-10 MeV to 20 bins:

```
NBINS = [5, 20]
ERANGE = [0.05, 1.5, 10.0]
```

MTRXFILE

Input string name for original, full resolution response matrix. Provided only for compatibility with previous versions.

NBINS

Scalar or vector specifying the number of bins into which to resample both the SDR and the corresponding response matrix. If the value is a scalar, then the entire original spectrum and response will be resampled into the number of bins specified. If the value is a vector, each element of the vector must have a corresponding energy range associated with it. The energy range for each element of NBINS will be provided by the keyword ERANGE.

NOFRACTIONS

If present and non-zero, the keyword NOFRACTIONS specifies that rebinning edges and widths be adjusted to coincide with integral channels of the initial data binning.

/NOLOAD

Input logical flag, which if present and nonzero, indicates that the common block should not be populated with the concatencated, re-binned matrix after rebinning. Default is to populate the common block.

OUTROOT

Input string defining the output root file name for the rebinned response matrix (if provided multiple spectra, then this matrix will also be concatenated). Provided only for compatibility with previous versions.

INMAP

Input array which provides alternate method for mapping input channels into output channels. There is one element in INMAP for every channel in the input SDR. Content of the element is the output channel # (0.. Nout-1). Content of less than zero indicates that the channel is not mapped to any output channel. INMAP usage requires /NOFRACTIONS specification as well. If INMAP is provided, parameters EDGES and WIDTHS are ignored on input, but return the edges and widths of the first SDR that are identified by the INMAP.

OUTMAP

Output array of the form of INMAP which indicates the binning that was performed. Valid only in conjunction with /NOFRACTIONS.

Discussion

This procedure resamples both the input SDR and its corresponding response matrix into the desired bin sizes. The preferred usage will read data from the input SDR and its parent SDB file, and write to the output output SDB file named by OUTSDB. The "mtrxfile" input and output are provided only for compatiblity. One or both of [OUTSDB, OUTROOT] must be provided.

NOTE: MTRX_REBIN will support response matrices containing more than one source per spectra only when input and output matrices are

accessed through an SDR, not with "mtrxfiles".

OSSE_COORDS

This procedure uses the coordinates of the spacecraft X- and Z-axes and the OSSE scan and trans-scan angles to determine the coordinates of the center of the collimator field-of-view. The format of this command is:

Calling Sequence

OSSE_COORDS, zra, zdc, xra, xdc, tscan, txscan, tra, tdc

[This procedure is written in IDL]

Arguments

ZRA
The input longitude-like sky coordinate of the spacecraft Z-axis, specified in degrees.

ZDC
The input latitude-like sky coordinate of the spacecraft Z-axis, specified in degrees.

XRA
The input longitude-like sky coordinate of the spacecraft X-axis, specified in degrees.

XDC
The input latitude-like sky coordinate of the spacecraft X-axis, specified in degrees.

TSCAN
An input angle, in degrees, specifying the scan angle of the detector from the spacecraft Z-axis.

TXSCAN
An input angle, in degrees, specifying the trans-scan angle of the detector from the spacecraft XZ-plane.

TRA
The output longitude-like sky coordinate of the center of the collimator field-of-view.

TDC
The output latitude-like sky coordinate of the center of the collimator field-of-view.

NOTE: **This routine does not incorporate the details of the OSSE internal alignment and alignment to the spacecraft. This introduces errors generally less than 0.1°.**

OSSE_PPKAREA

Calling Sequence

OSSE_PPKAREA, scan, xscan, energy, photopk

Arguments

SCAN
Input scan angle(s) to compute the response.

XSCAN
Input Xscan angle(s) to compute the response.

ENERGY
Input float array containing 1 or more energies at which photopeak response is to be calculated.

PHOTOPK
Output array of photopeak responses in cm^2. Array is (n_energy, n_ang).

Keywords

/VERBOSE
An input logical switch to turn on diagnostic/informational messages.

Discussion

The DRM software is accessed to get the weights, which are then applied to the photopeak data from a previously-extracted data file. SIDE EFFECTS: Will work under IDL only, however the user must be linked to the DRM linkimage and mcsdb_catalog. Although no actual MCSDB storage files are accessed, they need to be available because an error results when a user tries to initialize the DRM module without these files.

PLOT_SRC

Calling Sequence

PLOT_SRC, sdr, mdldef, plottitle

Arguments

SDR
Input SDR pointer array or subarray containing OSSE spectrum data, generally the output product of CNTS_TO_PHOTONS.

MDLDEF
Input model definition record (or record array) including best fit parameters. The model record is an IDL record of structure MODSTR.

PLOTTITLE
Input string array containing the title for each plot. One title is sufficient for most cases, but if the data has been concatenated, then provide more than one title for each detector or AU.

Keywords

NBINS
Scalar or vector specifying the number of bins into which to resample both the SDR array and the response matrix. If the value is a scalar, then the entire original spectrum and response will be resampled into the number of bins specified. If the value is a vector, each element of the vector must have a corresponding energy range associated with it. The energy range for each element of NBINS will be provided by the keyword parameter ENE_RNG.

ENE_RNG
Vector specifying the energy range for each NBIN element, if NBINS is a vector. This parameter is ignored if NBINS is a scalar. The number of elements in ENE_RNG must be one greater than the number of elements in NBINS. For example, to resample spectrum between 0.05-1.5 MeV to 5 bins and spectrum between 1.5-10 MeV to 20 bins:

```
NBINS = [5, 20]
ENE_RNG = [0.05, 1.5, 10.0]
```

MODELFILE
Name of file containing input model information.

II,IO,OI
Input which controls log-lin, lin-log, lin-lin scaling. The default is log-log scaling when none of these keywords are present.

TEXT
Input 2-D vector specifying position of model text. Must be x,y coordinates in normalized units ranging from 0.0 to 1.0.

COMMENT
Input string arrray of dimensions NSDR specifying comment for !p.subtitle. The default subtitle defines OSSE viewing period data.

DISMODE
Input specifying datum display mode, 0 =nominal, 1 = cross w/ bars, 2 = diamond.

PSYM, XRANGE, YRANGE
As in IDL.

NOMODEL
Input switch which, if present and non-zero, will suppress plotting model data and model text.

NORESIDUAL
Input switch which, if present and non-zero, will suppress plotting residual graph.

VIEWGRAPH
Input switch which, if present and non-zero, will increase character size of plot for viewgraphs.

Discussion

The procedure PLOT_SRC plots fitted OSSE spectrum data (provided in units of *photons/sec-cm*^{2}*-MeV*), and the photon model used in the fit. The OSSE spectrum data is obtained via an input SDR, which should be the output product of the CNTS_TO_PHOTONS.

The procedure accesses the model parameter data either from a stored model file or from an input model record (or record array), and generates the photon model which is plotted over the spectrum data. A description of the model used in the fit is also provided in the output.

The data can be rebinned according to the optional keyword arguments NBINS and ENE_RNG. When NBINS is not provided, then the data will not be rebinned and the full resolution data will be displayed.

One plot is generated for every record in the SDR.

Related Routines

See the related routines CNTS_TO_PHOTONS, DO_FOLDANAL, MODELSTRUC, BUILD-MODEL, MODEL_READ, and MODEL_WRITE.

POINTERS

This application creates an IDL variable containing IGORE record pointers for the specified IGORE record. The format of this command is:

Calling Sequence

POINTERS, r_pointers

Arguments

R_POINTERS
An IDL variable which, on return, will contain the IGORE record pointers. The name of this variable also specifies the name of the IGORE record for which the pointers are to be returned.

Discussion

This procedure returns in R_POINTERS the pointers for the specified IGORE record. The IGORE record is specified by the name of the R_POINTERS argument.

This procedure is useful if records were created within a procedure or function and were not returned as arguments, or if the IDL record pointer variable was inadvertently corrupted.

PSF$OPEN

This procedure provides a method for opening a PSF file for subsequent access by IGORE applications. The format of this command is:

Calling Sequence

PSF$OPEN, lun, file_name [, file_status [, read_only]]

Arguments

LUN
An integer specifying the logical unit number to which the PSF file is to be attached.

FILE_NAME
A character string specifying the name of the PSF file to be opened.

FILE_STATUS
An optional IDL character string variable which specifies one of following FORTRAN style file status values: "OLD", "NEW", "UNKNOWN", or "SCRATCH". The default is "OLD".

READ_ONLY
An optional IDL variable which specifies readonly access when set to a logical 'true'. If specified and not a logical 'true', the file is opened for read and write access. The default value is readonly.

Discussion

This routine provides a method for opening a PSF file for subsequent access by IGORE applications.

If the logical unit specified by LUN is already attached to a file, an error is issued and the application aborted.

PSF$WRITE

This procedure provides the ability to write a subset of the information stored in an SDR to a PSF file. The format of this command is:

Calling Sequence

PSF$WRITE, lun, r_sdr

Arguments

LUN
An integer specifying the logical unit number of the PSF file to which the data are to be written.

R_SDR
The SDR record pointer array or subarray from which the data are to be obtained.

Discussion

The PSF file must be opened using PSF$OPEN with write access prior to using this application.

PUTAUX

This procedure provides the ability to place a value into the specified field of an AUX record of an SDR. The format of this command is:

Calling Sequence

PUTAUX, auxkey, field_name, r_sdr, value [, index]

[This procedure is written in IDL]

Arguments

AUXKEY

An integer value specifying the AUX type into which VALUE is to placed. An AUX record of this type must be present in the SDR.

See the "Spectral Data Base Header Definition" document for a list of the supported AUX key values and the corresponding structure types.

FIELD_NAME

A character string containing the name of the field in the AUX record into which VALUE is to be placed. This argument is ignored if the AUX record is an indirect field.

R_SDR

An SDR pointer array or subarray containing the AUX record into which VALUE is to be placed.

VALUE

An IDL variable containing the value which is to be placed into the specified field of the AUX record.

INDEX

An optional integer parameter specifying the index element of the AUX record array into which VALUE is to be placed. This optional argument is only valid if the specified AUX type is a record array.

Keywords

/ALL

If present and non-zero, /ALL indicates that the IDL variable VALUE is to be placed in **all** AUX record elements of R_SDR.

Discussion

This procedure provides the ability to store information in an SDR AUX field if the AUX type is an indirect field, or in a specific field of an SDR AUX record if the AUX type is an indirect record. If all AUX fields in the SDR are already in use, an error is reported and the routine is aborted.

An error is also reported if an AUX type of "Class 2" is specified and does not already exist in the SDR. See Table 5.1 in Chapter 5 for the complete list of AUX types and classes. See PUTAUXREC for "Class 2" and "Class 3" AUX types.

An example of the usage of PUTAUX is given below:

```
Igore> SDR$CREATE, r_sdr, 2
Igore> CREREC, 'IGR$AUID', r_auid, 5
Igore> PUTFLD, 'TAG', r_auid, [1, 2, 3, 4, 5]
Igore> PUTAUXREC, !AUX.AUID, r_sdr(0), auxrec=r_auid(0:3)
Igore> PUTAUXREC, !AUX.AUID, r_sdr(1), auxrec=r_auid
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=1)
GetAux$Ge%IGR-I-IMDMSG, Using fill records...

           1           2           3           4           0
           6           7           8           9          10
Igore> PUTAUX, !AUX.AUID, 'TAG', r_sdr(0), [11, 12, 13, 14]
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=1)
GetAux$Ge%IGR-I-IMDMSG, Using fill records...

          11          12          13          14           0
           6           7           8           9          10
Igore> PUTAUX, !AUX.AUID, 'TAG', r_sdr(0), 1, /all
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=1)
GetAux$Ge%IGR-I-IMDMSG, Using fill records...

           1           1           1           1           0
           6           7           8           9          10
Igore> PUTAUX, !AUX.AUID, 'TAG', r_sdr, -1, /all
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=1)
GetAux$Ge%IGR-I-IMDMSG, Using fill records...

          -1          -1          -1          -1           0
          -1          -1          -1          -1          10
Igore> PUTAUX, !AUX.AUID, 'TAG', r_sdr(1), -2, /all
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=1)
GetAux$Ge%IGR-I-IMDMSG, Using fill records...

          -1          -1          -1          -1           0
          -2          -2          -2          -2          -2
Igore> PUTAUX, !AUX.AUID, 'TAG', r_sdr(0), 5, index=3
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=1)
GetAux$Ge%IGR-I-IMDMSG, Using fill records...

          -1          -1          -1           5           0
          -2          -2          -2          -2          -2
```

```
Igore> PUTAUX, !AUX.AUID, 'TAG', r_sdr, 9, index=3, /all
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr, dimsize=1)
GetAux$Ge%IGR-I-IMDMSG, Using fill records...

          -1           -1           -1            9            0
          -2           -2           -2            9           -2
```

Related Routines

Also see the related function PUTAUXREC.

PUTAUXREC

This procedure provides the ability to place an IGORE record into an AUX record field of an SDR. The format of this command is:

Calling Sequence

PUTAUXREC, auxkey, r_sdr

[This procedure is written in IDL]

Arguments

AUXKEY

An integer value specifying the AUX key into which the record is to be stored. If an AUX record of this type is already stored in the SDR, it will be overwritten; otherwise, the first free AUX slot will be used.

See the "Spectral Data Base Header Definition" document for a list of the supported AUX key values and the corresponding structure types.

R_SDR

An SDR pointer, or a single element of a pointer array, into which the AUX record is to be placed.

Keywords

AUXREC

An IGORE record or record array which is to be placed into the specified AUX field. In the case of "Class 2" AUX fields, this is the only keyword parameter which should be passed. In the case of "Class 3" AUX fields this parameter is optional, but a "Header" record is then required.

HDR

An IGORE record pointer which will be copied to the "HEADER" portion of a "Class 3" AUX field. If HDR is specified with no data, any existing AUX field of this type is completely erased and only the header portion replaces it (*e.g.*, you cannot use this routine to replace only the "HEADER" record of a "Class 3" AUX field without destroying existing data as well). If the AUX type is "Class 3" and HDR is not passed as a keyword parameter, a blank header will be written followed by any AUXREC data record(s) provided. See Chapter 5 for complete AUX type definitions.

Discussion

This routine copies an IGORE record or record array into an IGORE AUX record of an SDR. The AUXREC and HDR record pointers must be of the appropriate type for the parent record and the specified AUXKEY. If all AUX fields in the SDR are already in use, an error is reported and the routine is aborted. An example of the usage of PUTAUXREC is given below:

```
Igore> SDR$CREATE, r_sdr, 2
Igore> CREREC, 'IGR$AU_DEF', r_audef, 2
Igore> CREREC, 'IGR$AUID', r_auid, 5
Igore> PUTFLD, 'SRC_DT', r_audef(0), [0.0, 1.0, 2.0, 3.0]
Igore> PUTFLD, 'SRC_DT', r_audef(1), [4.0, 5.0, 6.0, 7.0]
Igore> PUTFLD, 'TAG', r_auid, [1, 2, 3, 4, 5]
Igore> PUTAUXREC, !AUX.AUDEF, r_sdr(0), auxrec=r_audef(0)
Igore> PUTAUXREC, !AUX.AUDEF, r_sdr(1), auxrec=r_audef(1)
Igore> PUTAUXREC, !AUX.AUID, r_sdr(0), auxrec=r_auid(0:3)
Igore> PUTAUXREC, !AUX.AUID, r_sdr(1), auxrec=r_auid
Igore> PRINT, GETAUX( !AUX.AUDEF, 'SRC_DT', r_sdr(0))
        0.000000      1.00000      2.00000      3.00000
Igore> PRINT, GETAUX( !AUX.AUDEF, 'SRC_DT', r_sdr(1))
         4.00000      5.00000      6.00000      7.00000
Igore> PRINT, GETAUX( !AUX.AUDEF, 'SRC_DT', r_sdr)
        0.000000      1.00000      2.00000      3.00000
         4.00000      5.00000      6.00000      7.00000
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr(0))
         1           2           3           4
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr(1))
         1           2           3           4          5
Igore> PUTFLD, 'TAG', r_auid, [6, 7, 8, 9, 10]
Igore> PUTAUXREC, !AUX.AUID, r_sdr(1), auxrec=r_auid
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr(0))
         1           2           3           4
Igore> PRINT, GETAUX( !AUX.AUID, 'TAG', r_sdr(1))
         6           7           8           9         10
```

Related Routines

Also see the related function PUTAUX.

PUTFLD

This application provides the ability to place a value into the specified field of an IGORE record or record array. The format of this command is:

Calling Sequence

PUTFLD, field_name, r_pointer, value

Arguments

FIELD_NAME

A character string containing the name of the field into which the value is to be placed.

R_POINTER

The IGORE record or record array containing the field into which the value is to be placed.

VALUE

An IDL variable containing the value which is to be placed into the specified field of the IGORE record.

Keywords

ALL

Optional keyword that when set indicates that the specified value is to be placed into specified field of every element of the IGORE record array. This is the preferred alternative to the ALL_FLAG argument.

Discussion

This routine places the value passed in VALUE into the specified field of the IGORE record or record array R_POINTER, provided the value can be converted into the same type as the field. Note that if the field specified by FIELD_NAME is an indirect access field, no type conversion is performed on VALUE and the type of the indirect access field may be changed.

When R_POINTER is an array, the default action is to match up the VALUE array with the record array. The right-most index of the VALUE array is paired with the R_POINTER element number. In this case, they must have the same number of elements.

An example of the usage of PUTFLD is given below:

The optional argument ALL_FLAG is still supported but the preferred method is to use the keyword ALL.

```
Igore> SDR$CREATE, r_sdr, 2
Igore> PUTFLD, 'DAT', r_sdr(0), findgen(10)
Igore> PUTFLD, 'DAT', r_sdr(1), findgen(20)
Igore> HELP, GETFLD( 'DAT', r_sdr(0))
<Expression>    FLOAT     = Array(10)
Igore> HELP, GETFLD( 'DAT', r_sdr(1))
<Expression>    FLOAT     = Array(20)
Igore> PUTFLD, 'DAT', r_sdr, indgen( 10, 2)
Igore> HELP, GETFLD( 'DAT', r_sdr(0))
<Expression>    INT       = Array(10)
Igore> HELP, GETFLD( 'DAT', r_sdr(1))
<Expression>    INT       = Array(10)
Igore> PUTFLD, 'DAT', r_sdr, findgen( 10, 3), /ALL
Igore> HELP, GETFLD( 'DAT', r_sdr(0))
<Expression>    FLOAT     = Array(10, 3)
Igore> HELP, GETFLD( 'DAT', r_sdr(1))
<Expression>    FLOAT     = Array(10, 3)
Igore> HELP, GETFLD( 'DAT', r_sdr)
<Expression>    FLOAT     = Array(10, 3, 2)
Igore> PUTFLD, 'HDR.DETID', r_sdr(0), 1
Putfld_AF%IGR-E-IMDMSG, Indirect record has not been created for first
parent pointer
% Execution halted at $MAIN$   (PUTFLD).
Igore> SDR$INIT_HDR, r_sdr
Igore> PRINT, GETFLD( 'HDR.DETID', r_sdr)
       0       0
Igore> PUTFLD, 'HDR.DETID', r_sdr(0), 1
Igore> PRINT, GETFLD( 'HDR.DETID', r_sdr)
       1       0
Igore> PUTFLD, 'HDR.DETID', r_sdr, [2, 3]
Igore> PRINT, GETFLD( 'HDR.DETID', r_sdr)
       2       3
Igore> PUTFLD, 'HDR.DETID', r_sdr, -1, /ALL
Igore> PRINT, GETFLD( 'HDR.DETID', r_sdr)
      -1      -1
Igore> HELP , GETFLD( 'HDR.CMDSTATE_ID', r_sdr)
<Expression>    STRING    = Array(2)
Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(0))

Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(1))

Igore> PUTFLD, 'HDR.CMDSTATE_ID', r_sdr(0), 'String#1'
Igore> PUTFLD, 'HDR.CMDSTATE_ID', r_sdr(1), 'String#2'
Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(0))
String#1
Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(1))
String#2
Igore> PUTFLD, 'HDR.CMDSTATE_ID', r_sdr, 'String#3', /ALL
Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(0))
String#3
Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(1))
String#3
```

```
Igore> PUTFLD, 'HDR.CMDSTATE_ID', r_sdr, ['String#4', 'String#5']
Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(0))
String#4
Igore> PRINT, GETFLD( 'HDR.CMDSTATE_ID', r_sdr(1))
String#5
```

QUALIFY_FILE

This function processes a file specification by adding missing directory and extension fields, as well as optionally verifying the existence of the file.

Calling Sequence

return = QUALIFY_FILE, inname, outname

Arguments

INNAME
Input character string containing file specification to be processed.

OUTNAME
Output character string to receive processed file specification.

Keywords

DIR
Optional character string containing a directory specification for the file. If INNAME does not include a directory, the directory specified with this keyword will be included in the OUTNAME specification.

EXT
Optional character string containing a file extension for the file. If INNAME does not include a file extension, the extension specified with this keyword will be included in the OUTNAME specification.

EXIST
An optional switch specifying that the existence of the file should be verified. A logical FALSE is returned if the file does not exist.

Discussion

This procedure processes an input file specification. If the input file specification is a logical name, the logical is translated. If not, missing directory and extension specifications will applied from the optional DIR and EXT keywords. In all cases the resulting file specification is returned in OUTNAME.

If the keyword EXIST is set, the existence of the file is checked. A logical FALSE is returned if the file does not exist, otherwise the return value is always TRUE.

QUICKLOOK

This procedure reads in spectra from specified files and generates plots of spectra over two different energy ranges. The format of this command is:

Calling Sequence

QUICKLOOK, qlfiles

[This procedure is written in IDL]

Arguments

QLFILES

A scalar or array string variable containing the name(s) of files from which to extract spectra for plotting. All spectra from each file are plotted and included in the summary average spectra.

Keywords

BANDEDG

Specifies the lower end of each energy band for rate vs. time plots. May be a scalar for a single band or an array in the case of multiple bands. Number of elements must agree with bandwid keyword. If not present, no rate histories are accumulated.

BANDWID

Specifies the band width of each energy band for rate vs. time plots. May be a scalar for a single band or an array in the case of multiple bands. Number of elements must agree with bandedg keyword. If not present, no rate histories are accumulated.

P1RNG

Specifies one range of channels for spectrum plots. It is a two-element array containing [low channel, high channel] of the range. If not present, a default range of [12,50] is used.

P2RNG

Specifies a second range of channels for spectrum plots. It is a two-element array containing [low channel, high channel] of the range. If not present, a default range of [12,255] is used.

TITLE

A string variable that gets prepended to the standard title of each plot. The standard title contains the collection interval for spectra and the energy band definition for rate histories. If not supplied, nothing is prepended.

/ONERANGE
Switch which, if set, specifies that only one range of channels is plotted (lorng, either as specified by keyword or default). This allows twice as many spectra per page.

STARTTIME
Allows user to specify a start time for spectra to process. Any spectra with a start of collection before this time will not be plotted or included in averages. Variable is a string containing time in the format 'yy/ddd hh:mm:ss.sss'. If hh:mm:ss.sss is left out, 00:00:00 is assumed. If not set, processing will start at the beginning of the file.

STOPTIME
Allows user to specify a stop time for spectra to process. Any spectra with an end of collection after this time will not be plotted or included in averages. Variable is a string containing time in the format 'yy/ddd hh:mm:ss.sss'. If hh:mm:ss.sss is left out, 00:00:00 is assumed. Note that this means that a stop time of '93/137' will exclude any intervals including time after 93/137 00:00:00. If 93/137 is desired to be included, set stoptime='93/138'. If not set, processing will continue until the end of the file.

DETS
Allows user to specify a subset of detectors for display. If not set, all detectors in file are displayed. Format is an integer array of detector numbers (detectors numbered from 1-4).

/NOSPEC
Switch that disables individual interval spectrum plots if set. If not set, spectra are plotted. Sum spectra are plotted in either case.

CHARSIZE, XCHARSIZE, YCHARSIZE
Have standard IDL meanings. These are effective for the full page plots only (i.e. summed spectra, multi-detector time histories).

NOT_TAGS
Input array of SDB tagwords which should be left out of analysis.

/IO
Input switch which, if present and nonzero, specifies the plot should be in lin-log space.

/OI
Input switch which, if present and nonzero, specifies the plot should be in log-lin space.

/BYDSA
Input switch which, if present and nonzero, specifies the plot should be daily spectra by DSA.

TARGNAME
Character string which specifies target name. Necessary only if the target is not in the planning catalog.

Discussion

This procedure reads in spectra from specified files and generates plots of spectra over two different energy ranges. It also, generates plots of difference rates over multiple energy bands vs. time. It plots summary spectra live-time averaged over all the spectra in the input files. Energy calibrations are derived from live-time averaged rgain values from the input spectra.

It is primarily for use as a quick way to examine DAY_SUMMER results, this program can also be used to look at two-minute differences and raw spectra.

READ_DATAPTS

This procedure reads ASCII formatted data from a file into a two dimensional IDL array. The format of this command is:

Calling Sequence

READ_DATAPTS, filename, out_var

Arguments

FILENAME

Input character string specifying the file from which to read the data.

OUT_VAR

Output variable to receive the data array read from the file.

Keywords

COMMENTS

Output variable which receives any and all comments in the file (ie. that which is preceded by a semicolon) concatenated together.

Discussion

This routine reads ASCII formatted data from a file into a two dimensional IDL array. The file is expect to contain the size of each dimension, one dimension per line, immediately preceding the actual data. The first dimension will specify the number of rows in the output variable, the second the number of columns.

An initial comment block may precede the array dimension records. Comment records must begin with a semicolon. A brief example of a compatible data file follows:

```
;
; This file contains example array data for READ_DATAPTS.PRO
;
3
4
1 2 3 5
2 3 5 8
5 8 13 21
```

READ_PLANCAT

This procedure reads the OSSE Target Database. The format of this command is:

Calling Sequence

READ_PLANCAT, cat

[This procedure is written in IDL]

Arguments

CAT
An output structure array of type PLANCAT containing information about each of the targets given in the OSSE Target database.

Discussion

This procedure provides access to the OSSE target Database, returning information about all of the targets listed in the database in the parameter CAT.

See the routines FIND_SOURCE and FIND_SOURCE_ID for methods of searching the variable CAT for a specified target.

RECINDX

This procedure returns the index into the IGORE parent record for the passed record pointer. The format of this command is:

Calling Sequence

index = RECINDX(r_pointer)

[This procedure is written in IDL]

Arguments

R_POINTER
The IGORE record pointer, or single element of a pointer array, for which the index into the parent record array is requested.

RECNAME

This application returns the IGORE record name of the passed IGORE record pointer. The format of this command is:

Calling Sequence

record_name = RECNAME(r_pointer)

Arguments

R_POINTER

The IGORE record pointer, or single element of a pointer array, for which the parent record name is requested.

REC_EXIST

This function returns a logical flag indicating whether the specified IGORE record exists. The format of this command is:

Calling Sequence

flag = REC_EXIST([variable])

Arguments

VARIABLE

An optional IDL variable which is to be tested to determine whether it points to a valid IGORE record. This argument is ignored if the NAME keyword is provided.

Keywords

NAME

A character string used to determine whether a valid IGORE record exists having the same name. Assumed to be no longer than 15 characters.

Discussion

This routine determines whether the parameter passed as VARIABLE points to a valid IGORE record. If the keyword NAME is specified, the IGORE record table is tested to determine whether an IGORE record exists having the same name. The name will be truncated if longer than 15 characters. The returned function value FLAG is a byte variable indicating whether an IGORE record exists having the specified name. If a record exists having the specified name, the returned value of FLAG will be 255 (true); otherwise, the returned value will be 0 (false).

RENAME_REC

This procedure is used to rename an existing IGORE record. The format of this command is:

Calling Sequence

RENAME_REC, r_old_record, r_new_record

Arguments

R_OLD_RECORD

An IDL variable containing the record pointers to the IGORE record to be renamed.

R_NEW_RECORD

The new record pointer or pointer array which will contain the renamed input IGORE record pointer(s).

Keywords

NAME

A character string specifying the name for the new IGORE record. Will be truncated if longer than 15 characters.

/UNIQUE

If present and non-zero, /UNIQUE forces the generation of a unique name for the new IGORE record. This keyword is ignored if the NAME keyword is present.

Discussion

This routine renames an existing IGORE record, returning the pointers for the renamed record in the variable R_NEW_RECORD. The record specified by R_OLD_RECORD must already exist. If neither of the keywords NAME or /UNIQUE are present, the name of the created record is taken from the name of the IDL variable passed as R_NEW_RECORD. The name is assumed to be only 15 characters. The input record pointer R_OLD_RECORD must be a contiguous array.

RESAMPLE

This procedure provides a method for creating a matrix which maps specified input channels into desired output bins. The format of this command is:

Calling Sequence

RESAMPLE, edg1, wid1, edg2, wid2, matrix, owid2

[This procedure is written in IDL]

Arguments

EDG1
An input array of the initial bin lower edges.

WID1
An input array of the initial bin widths.

EDG2
An input array of the desired bin lower edges.

WID2
An input array of the desired bin widths.

MATRIX
An output matrix, dimensioned (NEDG2, NEDG1) where NEDG1 is the number of elements in EDG1 and NEDG2 is the number of elements in EDG2, which maps the input bins into the output bins.

OWID2
An output array of the resultant bin widths. The values in OWID2 may be slightly different from those specified in WID2 due to possible data gaps in the input bins.

Keywords

RANGE_SWITCH
An input 2-element array which defines the energies at which binning changes from LOW to MEDIUM and from MEDIUM to HIGH spectral ranges. The default is to take the highest channel of each range (*e.g.*, use all of the LOW range, then the remaining portion of the MEDIUM range).

/NATURAL
If present and non-zero, /NATURAL turns off the warning message that the input edge definition is monotonic increasing.

Related Routines

Also see the related function MERGECHAN.

RESTORE_LOGS

This procedure reassigns logical names from the input structure file_logs, associating logical names and full file specifications.

The format of this command is:

Calling Sequence

RESTORE_LOGS, logmap

[This procedure is written in IDL]

Arguments

LOGMAP

A single or array of structures of type file_logs with tags of LOGNM and FULLNM associating a logical name string and a full file specification string.

Keywords

Discussion

Process logical names will be defined or redefined. Informational message notes will indicate redefinition of logical names.

Related Routines

See the related routine SELECT_FILES.

RESTORE_REC

This procedure is used to restore the IGORE records from a previous IGORE session which had been saved using the procedure SAVE_REC. The format of this command is:

Calling Sequence

RESTORE_REC [, save_file]

Arguments

SAVE_FILE
An optional character string specifying the name of an IGORE save file from which the records are to be restored. The specified file must have been previously created using the procedure SAVE_REC. If not passed, the default file IGORE_SAVE.IGR is used.

Discussion

The routine restores records which have been previously saved using the procedure SAVE_REC. If neither of the keywords are present, all of the records from the IGORE save file are restored. If there are any saved records for which there is a currently defined record having the same record name, a prompt is issued to determine whether the record should overwrite the existing record, be restored with a different record name, or not be restored.

IDL variables containing pointers to the restored IGORE records are generated with the same name as the IGORE records.

NOTE: This procedure may only be called from the main level of IGORE; it may not be used within any procedures or functions.

NOTE: Since IGORE records are stored as one-dimensional arrays, the restored IDL pointer arrays will also be created as one-dimensional arrays. The IDL procedure `REFORM` can be used to regenerate multi-dimensional record pointer arrays, if necessary.

NOTE: If the saved IGORE records were originally created with the /UNIQUE keyword, the IDL variables created as pointers to the restored records will have unpredictable names.

SAVE_REC

This procedure is used to save IGORE records in a file for later access. The format of this command is:

Calling Sequence

SAVE_REC [, file [, confirm]]

Arguments

FILE

A character string specifying the name of the IGORE save file into which the IGORE records will be saved. If this keyword is not specified, the default file IGORE_SAVE.IGR is used.

CONFIRM

If present and non-zero, CONFIRM causes a request to be issued before saving each IGORE record to confirm that the operation should be performed for that record. The default action is to not prompt the user.

Discussion

This routine provides the ability to save IGORE records in a file for subsequent access using the procedure RESTORE_REC. An example of the usage of SAVE_REC is given below:

```
Igore> SAVE_REC, file='test1.igr'
Igore> if !quiet eq. 1 then confirm = 0 else confirm = 1
Igore> SAVE_REC, file='test4.igr', confirm
```

SC_ATTITUDE

This procedure reads the OSSE Observing Plan Database, providing information about the GRO attitudes and OSSE targets. The format of this command is:

Calling Sequence

SC_ATTITUDE, att

[This procedure is written in IDL]

Arguments

ATT

An output structure array of type ATTSTRUC containing information about each of the spacecraft attitudes and the OSSE primary and secondary targets.

SC_COORDS

This procedure uses the coordinates of the spacecraft X- and Z-axes and the coordinates of a specified target to determine the OSSE scan and trans-scan angles for the target. The format of this command is:

Calling Sequence

SC_COORDS, zra, zdc, xra, xdc, tra, tdc, tscan, txscan

[This procedure is written in IDL]

Arguments

ZRA
The input longitude-like sky coordinate of the spacecraft Z-axis, specified in degrees.

ZDC
The input latitude-like sky coordinate of the spacecraft Z-axis, specified in degrees.

XRA
The input longitude-like sky coordinate of the spacecraft X-axis, specified in degrees.

XDC
The input latitude-like sky coordinate of the spacecraft X-axis, specified in degrees.

TRA
The input longitude-like sky coordinate of the target, specified in degrees.

TDC
The input latitude-like sky coordinate of the target, specified in degrees.

TSCAN
The output angle, in degrees, indicating the scan angle of the target from the spacecraft Z-axis.

TXSCAN
The output angle, in degrees, indicating the trans-scan angle of the target from the spacecraft XZ-plane.

NOTE: **This routine does not incorporate the details of the OSSE internal alignment and alignment to the spacecraft. This introduces errors generally less than 0.1°.**

SDB$CLOSE

This procedure closes an SDB file previously opened by an IGORE application (*e.g.* SDB$OPEN). The format of this command is:

Calling Sequence

SDB$CLOSE, lun

Arguments

LUN
An integer parameter specifying the logical unit number to close.

Keywords

FREE
If set, specifies that LUN should be returned to the pool of available logical units.

Discussion

This routine closes the logical unit specified by LUN. Optionally, this routine also frees the appropriate logical unit from the logical unit table. Note that this routine will **not** close files opened by IDL.

The preferred method for closing SDB files is using the command IGR$CLOSE.

SDB$CLOSEALL

This procedure closes all SDB files currently open for the IGORE session. The format of this command is:

Calling Sequence

SDB$CLOSEALL

Arguments

NONE

Discussion

This routine closes **all** currently opened SDB files. This routine also frees the appropriate logical unit from the logical unit table, where necessary. Note that this routine will **not** close files opened by IDL.

SDB$DELETE

This procedure provides the ability to delete the information stored in a selected tagword and auxtype of an SDB file. The format of this command is:

Calling Sequence

SDB$DELETE, lun, tag, auxtype

Arguments

LUN
An integer specifying the logical unit number of the SDB file from which the data are to be deleted.

TAG
A longword scalar specifying the tagword of the SDB records to be deleted.

AUXTYPE
A longword scalar specifying the aux type of the SDB records to be deleted.

Discussion

The SDB file must be opened using SDB$OPEN with the /WRITE and /OLD keywords prior to using SDB$DELETE.

If the number of array elements in the array TAG is less than or equal to zero, a warning is issued and no action is taken.

If a record cannot be deleted for any reason, a warning is issued and the routine continues on to the next tagword.

Deleting records does not actually free disk space for reuse. The DCL command $CONVERT creates a new file with the disk space released.

SDB$DIR

This procedure provides a method for performing a directory of records contained in the specified SDB file. The format of this command is:

Calling Sequence

SDB$DIR, lun

Arguments

LUN
An integer specifying the logical unit number of the SDB file for which the directory is to be performed.

Discussion

The file must have been opened (*e.g.*, using SDB$OPEN) prior to calling this routine. This application also provides the ability to write the file directory to an ASCII file.

SDB$HITAG

This function provides information about the highest valued primary (AUXTYPE = 0) tagword present in a specified SDB file. The format of this command is:

Calling Sequence

hightag = SDB$HITAG(lun)

Arguments

LUN
An integer specifying the logical unit number of the SDB file for which the tagword information is desired.

Discussion

This routine returns in HIGHTAG the value of the highest valued primary (AUXTYPE = 0) tagword in the specified file. The file must have been opened (*e.g.*, using SDB$OPEN) prior to calling this routine.

SDB$LOOK

This procedure provides a method for displaying the contents of a selected tagword contained in an SDB file. The format of this command is:

Calling Sequence

SDB$LOOK, lun

Arguments

LUN
An integer specifying the logical unit number of the SDB file for which the contents listing is to be displayed.

Discussion

The file must have been opened (*e.g.*, using SDB$OPEN) prior to calling this routine. The application prompts the user for the tagword to be displayed and provides the ability to write the listing to an ASCII file.

SDB$LOWTAG

This function provides information about lowest valued primary (AUXTYPE = 0) tagword present in a specified SDB file. The format of this command is:

Calling Sequence

lowtag = SDB$LOWTAG(lun)

Arguments

LUN

An integer specifying the logical unit number of the SDB file for which the tagword information is desired.

Discussion

This routine returns in LOWTAG the value of the lowest valued primary (AUXTYPE = 0) tagword in the specified file. The file must have been opened (*e.g.*, using SDB$OPEN) prior to calling this routine.

SDB$NTAG

This function provides information about the number of primary (AUXTYPE = 0) tagwords present in a specified SDB file. The format of this command is:

Calling Sequence

ntag = SDB$NTAG(lun)

Arguments

LUN
An integer specifying the logical unit number of the SDB file for which the tagword information is desired.

Discussion

This routine returns in NTAG the total number of primary (AUXTYPE = 0) tagwords in the specified file. The file must have been opened (*e.g.*, using SDB$OPEN) prior to calling this routine.

SDB$OPEN

This procedure provides a method for opening an SDB file for subsequent access by IGORE applications. The format of this command is:

Calling Sequence

SDB$OPEN, lun, file_name

Arguments

LUN

An integer specifying the logical unit number to which the SDB file is to be attached.

FILE_NAME

A character string specifying the name of the SDB file to be opened. The length of this parameter cannot exceed 20 characters. This argument may represent a logical pointing to the SDB file to be opened. By default SDB$OPEN opens files for readonly access.

Keywords

/CLOSE

If present and non-zero, /CLOSE specifies that the logical unit number or file is to be closed, if open, and re-opened using the specified logical unit number.

ERSTAT

A variable to receive the status of the open. A non-zero status code represents an an errror opening the SDB file. Zero indicates success.

/GET_LUN

If present and non-zero, /GET_LUN specifies that the logical unit number to be opened is to be obtained from the logical unit table and returned in LUN.

/NEW

If present and non-zero, /NEW indicates that a new SDB file is to be opened for write access.

/OLD

If present and non-zero, /OLD indicates that an existing SDB file is to be opened. Readonly access is the default for files opened with the /OLD keyword.

/WRITE

If present and non-zero, /WRITE specifies that the SDB file is to be opened with read/write access. /NEW is the default for files opened with the /WRITE keyword.

Discussion

This routine provides a method for opening an SDB file for subsequent access by IGORE applications. For existing files, the search for the SDB file specified by FILE_NAME is performed as described in Chapter 2.

If the logical unit specified by LUN is already attached to a file and the keyword /CLOSE is not specified, an error is issued and the application aborted.

Note that the argument FILE_NAME cannot exceed 20 characters; for file names longer than 20 characters a logical should be used.

Related Routines

See the related routine SELECT_FILES.

SDB$READ

This procedure provides a method of reading selected data from an SDB file into an IGORE SDR. The format of this command is:

Calling Sequence

SDB$READ, lun, r_sdr, tag

Arguments

LUN
An integer specifying the logical unit number from which the data are to be read.

R_SDR
An SDR pointer array or subarray into which the data are to be placed.

TAG
A longword scalar or array specifying the tagwords to be read. For non-positive values of TAG, a sequential read is performed and the tagword of the record read is returned in TAG, as well as in the TAG field of the corresponding element of R_SDR. Note that, since IDL passes subarrays as temporary arrays, if a subarray is passed as TAG, sequential reads will still be performed but the non-positive elements of TAG will **not** be modified to represent the actual tagwords read.

Keywords

AUX1
A longword integer specifying the AUX type(s) to be loaded into the SDR from the SDB file. The parameter AUX1 is a bit-pick in which bits 0–31 represent AUX types #0–31. Only the AUX types which are actually available in the SDB file are loaded into the SDR. This argument can be used as a shorthand method of specifying AUX data fields. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUX2
A longword integer specifying the AUX type(s) to be loaded into the SDR from the SDB file. The parameter AUX2 is a bit-pick in which bits 0–31 represent AUX types #32–63. Only the AUX types which are actually available in the SDB file are loaded into the SDR. This argument can be used as a shorthand method of specifying AUX data fields. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUXA
A two element array of longwords, the first element of which functions identically to

AUX1, the second functioning identically to AUX2. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

CALRPHA
An integer specifying the value of the SDB header variable CALRPHA for the SDB record to be loaded into the SDR. The default value of CALRPHA is −1, indicating no selection on CALRPHA is to be performed.

DETID
An integer indicating the detector(s) which are to be loaded into the SDR. The parameter DETID is a bit-pick (like the SDB header variable DETID) in which bits 0–3 represent detectors #1–4, respectively.

DPTBL
A longword integer specifying the value of the SDB header variable DPTBL for the SDB record to be loaded into the SDR. The default value of DPTBL is −1, indicating no selection on DPTBL is to be performed.

ERSTAT
A variable to receive an array of status codes, one status code element per SDR element. A non-zero status code represents an an errror retrieving at least one of the requested fields for that element. A status code value of -1 in the first element of the array indicates that no data was read for any element.

/NOHDR
If present and non-zero, /NOHDR indicates that the SDB header is **not** to be loaded into the SDR.

/REW
If present and non-zero, /REW indicates that the read should begin from the first record in the file.

STATUS
The name of an IDL variable which, if present, upon return contains the status of the SDB$READ operation. This variable can be used to determine whether SDB$READ completed successfully. A value of 0 indicates that no error occurred and the SDB file was read successfully.

TARGID
An integer specifying the value of the SDB header variable TARGID for the SDB record to be loaded into the SDR. The default value of TARGID is −1, indicating no selection on TARGID is to be performed.

/VERBOSE
If present and non-zero, /VERBOSE indicates that "per element" detail messages should be printed. By default these messages are not printed.

Auxiliary Data Keywords - AUDEF AUID BKG CALIB CEN CRC DAT DBKG DCEN DEDG DIR DWID EDG ERR MCDB1 MCDB2 MSDAT MSDSF MSERR MSSYERR NERR NMSERR NMSYERR NOTE NSY-ERR ORBHIST PRV PSDDEFF PSDEFF PSDMOD RELSPEC RERR RES RINFO RMDEF RMTRX RMVEC SPECLST SYERR WID
SDB$READ supports the full set of mnemonic AUX data selection keywords. Each keyword functions as a switch indicating that the corresponding AUX data field should be loaded. Only the AUX types which are actually available in the SDB file are loaded into the SDR. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

Discussion

The SDB file must be opened (*e.g.*, using SDB$OPEN) prior to using this application.

Unless otherwise specified, only the HDR field will be read into R_SDR. All indirect fields are freed prior to reading the data into the SDR. The FILE and TAG fields of R_SDR are modified to contain the SDB filename and tagword corresponding to the data which were read from the SDB file.

If the number of array elements in the arrays R_SDR and TAG are different, a warning is issued and only the first N elements are read, where N is the minimum number of elements in the R_SDR and TAG arrays.

If a tagword specified by TAG does not exist, a warning is issued unless /NOWARN is specified and the execution continued. The element of TAG producing the error is set to zero and the data associated with the corresponding element of R_SDR are deleted.

Related Routines

See the related routine SDR$GET.

SDB$READALL

This procedure provides a method of reading all the spectral data from an SDB file into an IGORE SDR. The format of this command is:

Calling Sequence

SDB$READALL, lun, r_sdr

Arguments

LUN

An integer specifying the logical unit number from which the data are to be read.

R_SDR

The output-only IGORE SDR array into which the data are to be placed. Name is assumed to be no longer than 15 characters.

Keywords

AUX1

A longword integer specifying the AUX type(s) to be loaded into the SDR from the SDB file. The parameter AUX1 is a bit-pick in which bits 0–31 represent AUX types #0–31. Only the AUX types which are actually available in the SDB file are loaded into the SDR. This argument can be used as a shorthand method of specifying AUX data fields. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUX2

A longword integer specifying the AUX type(s) to be loaded into the SDR from the SDB file. The parameter AUX2 is a bit-pick in which bits 0–31 represent AUX types #32–63. Only the AUX types which are actually available in the SDB file are loaded into the SDR. This argument can be used as a shorthand method of specifying AUX data fields. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUXA

A two element array of longwords, the first element of which functions identically to AUX1, the second functioning identically to AUX2. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

CALRPHA

An integer specifying the value of the SDB header variable CALRPHA for the SDB record to be loaded into the SDR. The default value of CALRPHA is −1, indicating no selection on CALRPHA is to be performed.

DETID
An integer indicating the detector(s) which are to be loaded into the SDR. The parameter DETID is a bit-pick (like the SDB header variable DETID) in which bits 0–3 represent detectors #1–4, respectively.

DPTBL
A longword integer specifying the value of the SDB header variable DPTBL for the SDB record to be loaded into the SDR. The default value of DPTBL is −1, indicating no selection on DPTBL is to be performed.

/NOHDR
If present and non-zero, /NOHDR indicates that the SDB header is **not** to be loaded into the SDR.

/REW
If present and non-zero, /REW indicates that the read should begin from the first record in the file.

STATUS
The name of an IDL variable which, if present, upon return contains the status of the SDB$READ operation. This variable can be used to determine whether SDB$READ completed successfully. A value of 0 indicates that no error occurred and the SDB file was read successfully.

TARGID
An integer specifying the value of the SDB header variable TARGID for the SDB record to be loaded into the SDR. The default value of TARGID is −1, indicating no selection on TARGID is to be performed.

/UNIQUE
If present and non-zero, /UNIQUE forces the generation of a unique name for the output IGORE record.

/VERBOSE
If present and non-zero, /VERBOSE indicates that "per element" detail messages should be printed. By default these messages are not printed.

Auxiliary Data Keywords - AUDEF AUID BKG CALIB CEN CRC DAT DBKG DCEN DEDG DIR DWID EDG ERR MCDB1 MCDB2 MSDAT MSDSF MSERR MSSYERR NERR NMSERR NMSYERR NOTE NSYERR ORBHIST PRV PSDDEFF PSDEFF PSDMOD RELSPEC RERR RES RINFO RMDEF RMTRX RMVEC SPECLST SYERR WID
SDB$READ supports the full set of mnemonic AUX data selection keywords. Each keyword functions as a switch indicating that the corresponding AUX data field should be loaded. Only the AUX types which are actually available in the SDB file are loaded into the SDR. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

Discussion

The SDB file must be opened (*e.g.* using SDB$OPEN) prior to using this application.

Unless otherwise specified, only the HDR field will be read into R_SDR. All indirect fields are freed prior to reading the data into the SDR. The FILE and TAG fields of R_SDR are modified to contain the SDB filename and tagword corresponding to the data which were read from the SDB file.

If R_SDR does not exist prior to the call to SDB$READALL, it will be created. If the record R_SDR does exist, it will be deleted and recreated with the proper dimension size. The maximum number of records that can be read by SDB$READALL is 3000. If more than 3000 records are to be read, R_SDR must be created with a sufficient number of elements prior to the call to SDB$READALL.

NOTE: The keywords are the same as SDB$READ, with the exception that SDB$READALL does not support the ERSTAT keyword.

SDB$REWRITE

This procedure provides the ability of rewriting the information stored in an SDR to the original SDB file. The format of this command is:

Calling Sequence

SDB$REWRITE, lun, r_sdr

Arguments

LUN

An integer specifying the logical unit number of the SDB file to which the data are to be written.

R_SDR

The SDR pointer array or subarray from which the data are to be obtained.

Discussion

The SDB file must be opened using SDB$OPEN with the /WRITE keyword prior to using SDB$REWRITE.

The tagwords used to rewrite the records is specified in the TAG field of the SDR.

NOTE: This procedure currently supports only the rewriting of SDB headers.

NOTE: The SDB header field NCHAN must not be modified prior to the SDB$REWRITE operation, otherwise, results are unpredictable.

SDB$WRITE

This procedure provides the ability to write the information stored in an SDR to a selected tagword of an SDB file. The format of this command is:

Calling Sequence

SDB$WRITE, lun, r_sdr, tag

Arguments

LUN

An integer specifying the logical unit number of the SDB file to which the data are to be written.

R_SDR

The SDR pointer array or subarray from which the data are to be obtained.

TAG

A longword scalar or array specifying the tagwords of the SDB records to be written. For non-positive values of TAG, a sequential write is performed and the tagword of the record written is returned in TAG. Note that, since IDL passes subarrays as temporary arrays, if a subarray is passed as TAG sequential writes will be performed but the non-positive TAG elements will **not** be modified to represent the actual tagwords written.

Keywords

AUX1

A longword integer specifying the AUX type(s) to be written to the SDB file from the SDR. The parameter AUX1 is a bit-pick in which bits 0–31 represent AUX types #0–31. Only the AUX types which are actually available in the SDR are written to the SDB file. This argument is a shorthand alternative to specifying individual AUX type keywords. See Table 5.1 in Chapter 5 for the complete list of AUX types, desciptions, related structures, and corresponding keywords.

AUX2

A longword integer specifying the AUX type(s) to be written to the SDB file from the SDR. The parameter AUX2 is a bit-pick in which bits 0–31 represent AUX types #32–63. Only the AUX types which are actually available in the SDR are written to the SDB file. This argument is a shorthand alternative to specifying individual AUX type keywords. See Table 5.1 in Chapter 5 for the complete list of AUX types, desciptions, related structures, and corresponding keywords.

AUXA
A two element array of longwords, the first element of which functions identically to AUX1, the second functioning identically to AUX2. See Table 5.1 in Chapter 5 for the complete list of AUX types, desciptions, related structures, and corresponding keywords.

/HDR
If present and non-zero, /HDR indicates that the SDB header is to be written to the SDB file. This is the default mode.

/ALL
If present and non-zero, /ALL indicates that all memory resident aux records for the specified SDR are to be written to the SDB file. This is the default.

/WRITE_ALL
Same as /ALL.

Auxiliary Data Keywords - AUDEF AUID BKG CALIB CEN CRC DAT DBKG DCEN DEDG DIR DWID EDG ERR MCDB1 MCDB2 MSDAT MSDSF MSERR MSSYERR NERR NMSERR NMSYERR NOTE NSYERR ORBHIST PRV PSDDEFF PSDEFF PSDMOD RELSPEC RERR RES RINFO RMDEF RMTRX RMVEC SPECLST SYERR WID.
SDB$WRITE supports the full set of mnemonic AUX data selection keywords. Each keyword functions as a switch indicating that the corresponding AUX data field should be written. Only the AUX types which are actually available in the SDB file are written to the SDB file. See Table 5.1 in Chapter 5 for the complete list of AUX types, desciptions, related structures, and corresponding keywords.

NOTE: **The default is to write all memory resident aux records to the SDB file. Using specific aux keywords overrides the default and only the requested aux records are written provided they reside in memory.**

Discussion

The SDB file must be opened using SDB$OPEN with the /WRITE keyword prior to using SDB$WRITE.

Unless overridden with /NOHDR, the HDR field is always selected. For example, SDB$WRITE, lun, r_sdr, tag, /DAT writes the HDR field and well as the DAT field.

If the number of array elements in the arrays R_SDR and TAG are different, a warning is issued and only the first N elements are written, where N is the minimum number of elements in the R_SDR and TAG arrays.

For non-positive values of TAG, the next tagword in sequence is written and the tagword of the record written is returned in TAG. Note that, since IDL passes subarrays as temporary arrays, if a subarray is passed as TAG the next tagword in sequence will be written but the non-positive TAG elements will **not** be modified to represent the actual tagwords written.

If a keyword flag for a field is specified and the field is not available in the SDR, this routine does not attempt to read the requested information from the original SDB file.

The FILE and TAG fields of R_SDR are undefined prior to the call to SDB$WRITE, they are modified to reflect the SDB filename and tagword to which the data were written.

If the record(s) already exist, SDB$WRITE will not rewrite the record(s). Instead the following message will be issued:

```
SDB$WRITE%IGR-W-IMDMSG, Error writing TAG #n to LUN #m.

  Status = 001A0032
```

Related Routines

See the related routines SDB$REWRITE and SDB$DELETE.

SDR$ADD

This procedure performs a channel-by-channel unweighted addition of the spectra stored in the two input SDR's, storing the result in an output SDR. The format of this command is:

Calling Sequence

SDR$ADD, r_sdr1, r_sdr2, r_out

Arguments

R_SDR1
The first of two input SDR pointer arrays or subarrays for which the spectral addition is to be performed.

R_SDR2
The second of two input SDR pointer arrays or subarrays for which the spectral addition is to be performed.

R_OUT
The output SDR pointer array or subarray into which the resulting summed spectra are to be placed.

Keywords

/LTNORM
If present and non-zero, /LTNORM specifies the input spectra are to be live-time normalized prior to performing the spectral addition. The default action is to **not** perform live-time normalization of the input spectra. See the discussion below for more details.

SPECUNIT
An integer specifying the spectrum units into which the input spectra are to be converted prior to the spectral addition. The spectrum units are specified in the same way as the SDB header variable SPECUNIT. The default value of SPECUNIT is 0, indicating the input spectral units are not to be modified.

MODE
An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 0, indicating that if any indirect fields in the input SDRs are not available, they will be read from the corresponding SDB file and stored in a temporary area.

XSCALE
A real variable specifying a scale factor by which the spectra in R_SDR2 are multi-

plied prior to the spectral addition. The default value of XSCALE is 1.0.

Discussion

This routine performs the spectral addition for each element of the input pointer arrays R_SDR1 and R_SDR2, storing the resultant spectrum in the output record R_OUT. Any data stored in R_OUT will be overwritten. The SDB header for the output spectrum is constructed from the two input headers as described in Chapter 5. The input spectra are not affected.

If SPECUNIT is not specified, the spectrum units are determined from the first spectrum of R_SDR1.

If the units of both of the input spectra are COUNTS, and the flag /LTNORM is not specified, the spectral addition is performed directly on the raw count spectra without live-time normalization. The resulting spectrum represents the arithmetic sum of the two input spectra and is given by:

$$Sum(i) = Spec1(i) + XSCALE \times Spec2(i)$$

$$Err(i) = \sqrt{Err1(i)^2 + (XSCALE \times Err2(i))^2}$$

where i is the spectrum channel number. The units of the output spectrum will be COUNTS.

If /LTNORM is provided, or if the units of both of the input spectra are already live-time normalized, the resulting spectrum represents the unweighted mean of the two input spectra and is given by:

$$Sum(i) = \frac{1}{1 + XSCALE}(Spec1(i) + XSCALE \times Spec2(i))$$

$$Err(i) = \frac{1}{1 + XSCALE}\sqrt{Err1(i)^2 + (XSCALE \times Err2(i))^2}$$

In cases where an error condition occurs, all information stored in R_OUT associated with the corresponding spectrum element is deleted and processing is continued with the next valid record element.

SDR$CREATE

This procedure creates an IGORE SDR array having the specified name and number of elements. The format of this command is:

Calling Sequence

SDR$CREATE, r_sdr, ndims

Arguments

R_SDR

The IDL variable into which the IGORE SDR pointers will be placed. The IGORE record created by this routine will be of type IGR$SDR. The name will be truncated if longer than 15 characters.

NDIMS

An integer specifying the number of elements in the record array to be created.

Keywords

NAME

A character string which, if present, specifies the name of the IGORE record to be created. The name will be truncated if longer than 15 characters. If not provided, the IGORE record will be created using the name of the IDL variable passed as R_SDR.

/UNIQUE

If present and non-zero, /UNIQUE forces the generation of a unique name for the IGORE record. Ignored if NAME is specified.

Discussion

This routine creates an IGORE record array having a structure type IGR$SDR. Note that SDR$CREATE performs the same operation as the routine CREREC with the structure type IGR$SDR. An example of the usage of SDR$CREATE is given below:

```
Igore> SDR$CREATE, r_sdr1, 10, name='sdr1'
Igore> SDR$CREATE, r_sdr2, 10
Igore> CREREC, 'IGR$SDR', r_sdr3, 10, name='sdr3'
Igore> CREREC, 'IGR$SDR', r_sdr4, 10
```

NOTE: IGORE does not currently support record arrays of more than one dimension.

SDR$FREE

This procedure deletes selected fields from an SDR and, in the case of indirect records or fields, frees the associated VM. The format of this command is:

Calling Sequence

SDR$FREE, r_sdr

Arguments

R_SDR

A SDR pointer array or subarray for which selected fields are to be deleted.

Keywords

AUX1

An integer or integer array specifying the AUX type(s) to be deleted from the SDR. AUX1 is a bit-pick in which bits 0–31 represent AUX data types #0–31. This argument is a shorthand alternative to specifying individual AUX type keywords. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUX2

An integer or integer array specifying the AUX type(s) to be deleted from the SDR. AUX2 is a bit-pick in which bits 0–31 represent AUX data types #32–63. This argument is a shorthand alternative to specifying individual AUX type keywords. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUXA

A two element array of longwords, the first element of which functions identically to AUX1, the second functioning identically to AUX2. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

/ALL

If present and non-zero, /ALL indicates that **all** indirect fields in the SDR are to be deleted, preserving only the "FILE" and "TAG" fields. This is the default action if no keywords are provided.

/FILE

If present and non-zero, /FILE indicates that the "FILE" fields in the SDR are to be cleared.

/HDR

If present and non-zero, /HDR indicates that the SDB header is to be deleted from the SDR.

/TAG

If present and non-zero, /TAG indicates that the "TAG" fields in the SDR are to be zeroed.

Auxiliary Data Keywords - AUDEF AUID BKG CALIB CEN CRC DAT DBKG DCEN DEDG DIR DWID EDG ERR MCDB1 MCDB2 MSDAT MSDSF MSERR MSSYERR NERR NMSERR NMSYERR NOTE NSYERR ORBHIST PRV PSDDEFF PSDEFF PSDMOD RELSPEC RERR RES RINFO RMDEF RMTRX RMVEC SPECLST SYERR WID.

SDR$FREE supports the full set of mnemonic AUX data selection keywords. Each keyword functions as a switch indicating that the corresponding AUX data field should be freed. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

Discussion

This procedure provides the ability to delete selected fields from an SDR. If the specified field is an indirect record or indirect field, the associated VM is deallocated.

If no keywords are provided, **all** indirect fields and indirect records in the SDR are deleted and the associated VM deallocated. If any keywords are present, only the specified fields will be deleted.

SDR$GET

This procedure provides a method of reading data stored in SDB files specified by the FILE and TAG fields stored in the passed SDR. The format of this command is:

Calling Sequence

SDR$GET, r_sdr

Arguments

R_SDR

An SDR pointer array or subarray containing the SDB filenames to open and the tagwords to be read and into which the data are to be placed.

Keywords

AUX1

A longword integer specifying the AUX type(s) to be loaded into the SDR from the SDB file. The parameter AUX1 is a bit-pick in which bits 0–31 represent AUX types #0–31. Only the AUX types which are actually available in the SDB file are loaded into the SDR. This argument can be used as a shorthand method of specifying AUX data fields. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUX2

A longword integer specifying the AUX type(s) to be loaded into the SDR from the SDB file. The parameter AUX2 is a bit-pick in which bits 0–31 represent AUX types #32–63. Only the AUX types which are actually available in the SDB file are loaded into the SDR. This argument can be used as a shorthand method of specifying AUX data fields. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

AUXA

A two element array of longwords, the first element of which functions identically to AUX1, the second functioning identically to AUX2. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

CALRPHA

An integer specifying the value of the SDB header variable CALRPHA for the SDB record to be loaded into the SDR. The default value of CALRPHA is −1, indicating no selection on CALRPHA is to be performed.

DETID
An integer indicating the detector(s) which are to be loaded into the SDR. The parameter DETID is a bit-pick (like the SDB header variable DETID) in which bits 0–3 represent detectors #1–4, respectively.

DPTBL
A longword integer specifying the value of the SDB header variable DPTBL for the SDB record to be loaded into the SDR. The default value of DPTBL is −1, indicating no selection on DPTBL is to be performed.

ERSTAT
A variable to receive an array of status codes, one status code element per SDR element. A non-zero status code represents an an errror retrieving at least one of the requested fields for that element. A status code value of -1 in the first element of the array indicates that no data was read for any element.

/NOHDR
If present and non-zero, /NOHDR indicates that the SDB header is **not** to be loaded into the SDR.

/REW
If present and non-zero, /REW indicates that the read should begin from the first record in the file.

STATUS
The name of an IDL variable which, if present, upon return contains the status of the SDB$READ operation. This variable can be used to determine whether SDB$READ completed successfully. A value of 0 indicates that no error occurred and the SDB file was read successfully.

TARGID
An integer specifying the value of the SDB header variable TARGID for the SDB record to be loaded into the SDR. The default value of TARGID is −1, indicating no selection on TARGID is to be performed.

/VERBOSE
If present and non-zero, /VERBOSE indicates that "per element" detail messages should be printed. By default these messages are not printed.

Auxiliary Data Keywords - AUDEF AUID BKG CALIB CEN CRC DAT DBKG DCEN DEDG DIR DWID EDG ERR MCDB1 MCDB2 MSDAT MSDSF MSERR MSSYERR NERR NMSERR NMSYERR NOTE NSYERR ORBHIST PRV PSDDEFF PSDEFF PSDMOD RELSPEC RERR RES RINFO RMDEF RMTRX RMVEC SPECLST SYERR WID
SDR$GET supports the full set of mnemonic AUX data selection keywords. Each keyword functions as a switch indicating that the corresponding AUX data field should be loaded. Only the AUX types which are actually available in the SDB file are loaded into the SDR. See Table 5.1 in Chapter 5 for the complete list of AUX types, descriptions, related structures, and corresponding keywords.

Discussion

The FILE and TAG fields of R_SDR are used to define the SDB file and tagword to be read. If either of these fields is invalid (*e.g.* no filename or a non-existent tagword), a warning message is issued, all information stored in R_SDR associated with the offending spectrum is deleted, and processing is continued with the next valid record element.

For valid FILE and TAG fields, the specified data are read from the SDB file and placed into the appropriate element of R_SDR. The SDB files are opened and closed, as required.

Note that SDR$GET opens any required SDB files, and these files may remain open after the completion of SDR$GET. The routine IGR$CLOSE may be used to close any files opened by SDR$GET.

SDR$GETFLD

This function returns data stored in an IGORE record or record array into a local IDL variable. The format of this command is:

Calling Sequence

return = SDR$GETFLD(field_name, r_pointers [, dimsize [, mode]])

Arguments

FIELD_NAME

A character string specifying the IGORE structure field name of the data to be extracted.

R_POINTERS

An IGORE record array or subarray for which the specified field data are to be extracted.

Keywords

DIMSIZE

If present and zero, DIMSIZE specifies that the dimensions of the output IDL variable should be determined from the first field in the record array R_POINTERS. This keyword is only valid when accessing an indirect field. Default is to dimension the output IDL variable from the size of the largest in the record array R_POINTERS.

MODE

An integer specifying the method of accessing the requested indirect field or record. See Table 2.1. The default value of MODE is 0.

Discussion

This procedure has the same functionality as GETFLD with the added feature that, if R_POINTERS is an SDR, the requested information can be retrieved directly from the associated SDB file without having been previously read into memory. This feature has the effect of minimizing the amount of memory required when dealing with large numbers of SDRs. Note that there is a time performance penalty when accessing the information from an SDB file.

SDR$GET_FIT

This procedure provides a method of evaluating a fit function defined in an FDR using the EDG field from the SDR and storing the resulting values in the DAT field of the SDR. The format of this command is:

Calling Sequence

SDR$GET_FIT, r_fdr, r_sdr

Arguments

R_FDR

An FDR pointer array or subarray containing the fit definitions.

R_SDR

An SDR pointer array or subarray containing the channel information used to evaluate the fit function.

Keywords

MODE

An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 0, indicating that if any indirect fields in the input SDRs are not available, they will be read from the corresponding SDB file and stored in a temporary area.

Discussion

This routine uses the fit function defined in R_FDR and evaluates the function using the EDG field of R_SDR. Note that the EDG field represents the lower edge of the channel. The resulting function values are stored in the DAT field of R_SDR, overwriting the original DAT field, if present.

If a WID field is available in the SDR, it is used to determine the upper edges of the channels; otherwise, the channels are assumed to be contiguous and the width of channel i is determined from the lower edges of channels i and $i+1$.

For fit models which do not integrate over the channels, the function is evaluated at the channel centers. If the channel edge array is not available in R_SDR, the lower channel edges are set to integer values starting at zero.

SDR$INIT_HDR

This procedure initializes the SDB header, FILE and TAG fields of an SDR. The format of this command is:

Calling Sequence

SDR$INIT_HDR, r_sdr

Arguments

R_SDR

An SDR pointer array or subarray for which the initialization is to be performed.

HDR

An optional scalar pointer to an IGORE SDB header record of type IGR$SDB_HDR which will be used to initialize the HDR fields of R_SDB.

Discussion

The FILE and TAG fields of the SDR are initialized to " " and 0, respectively. If an SDB header exists in R_SDR, it is deleted and the associated VM deallocated. If the optional parameter HDR is provided, the SDB header record pointed to will be copied into the indirect HDR record of each element of R_SDR; otherwise, a separate SDB header will be created for each element of R_SDR, with only minimal initialization of the SDB header variables.

SDR$LIST

This procedure provides a listing of the values of the fields for the passed SDR. The format of this command is:

Calling Sequence

SDR$LIST, r_sdr

Arguments

R_SDR
An SDR pointer array or subarray for which a listing of the fields is requested.

Discussion

The listing includes the element number, the record name, the FILE field, the TAG field, and the record pointers for each of the indirect fields in the SDR.

This routine can be used to determine which fields of an SDR have been populated. A zero in one or both of the address fields indicates that the indirect record or field has not been read into the SDR. An example of the usage of SDR$LIST is given below:

```
Igore> file = SELECT_FILES( '$prod:p913120000_spm.sdb')
Igore> SDB$OPEN, 1, file
Igore> SDR$CREATE, r_sdr, 5
Igore> SDB$READ, 1, r_sdr, LONARR( 5), /hdr, /dat
Igore> SDR$LIST, r_sdr(1:2)
Record name: R_SDR, Element #0
File Name: P913120000_SPM
Tag =  102

******  VM Pointer Information  ******
HDR Addresses    -  6984896,  6984950
DAT Addresses    -  6985992,  6986033     Key =    0
ERR Addresses    -        0,        0     Key =   -1
EDG Addresses    -        0,        0     Key =   -1
AUX(0) Addresses -        0,        0     Key =   -1
AUX(1) Addresses -        0,        0     Key =   -1
AUX(2) Addresses -        0,        0     Key =   -1
AUX(3) Addresses -        0,        0     Key =   -1
AUX(4) Addresses -        0,        0     Key =   -1
AUX(5) Addresses -        0,        0     Key =   -1
AUX(6) Addresses -        0,        0     Key =   -1
AUX(7) Addresses -        0,        0     Key =   -1
Flags, Seq. Id   -        0,        0
```

```
Dimensions      -      544,       0
Self Pointers   -  5384416,  5384617


Record name: R_SDR, Element #1
File Name: P913120000_SPM
Tag =  103

******  VM Pointer Information  ******
HDR Addresses   -  6988224,  6988278
DAT Addresses   -  6989320,  6989361     Key =    0
ERR Addresses   -        0,        0     Key =   -1
EDG Addresses   -        0,        0     Key =   -1
AUX(0) Addresses -       0,        0     Key =   -1
AUX(1) Addresses -       0,        0     Key =   -1
AUX(2) Addresses -       0,        0     Key =   -1
AUX(3) Addresses -       0,        0     Key =   -1
AUX(4) Addresses -       0,        0     Key =   -1
AUX(5) Addresses -       0,        0     Key =   -1
AUX(6) Addresses -       0,        0     Key =   -1
AUX(7) Addresses -       0,        0     Key =   -1
Flags, Seq. Id  -        0,        0
Dimensions      -      544,        0
Self Pointers   -  5384416,  5384764
```

Related Routines

See the related routine LISTREC, which produces a listing of both the direct and indirect records and fields in an IGORE record.

SDR$NORM

This procedure performs normalization of spectra stored in an SDR. The format of this command is:

Calling Sequence

SDR$NORM, r_sdr

Arguments

R_SDR
An SDR pointer array or subarray containing the spectra to be normalized.

Keywords

MODE
An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 1, indicating that if any indirect fields in the input SDR are not available, they will be read from the corresponding SDB file and stored in the SDR.

SPECUNIT
An integer specifying the spectrum units into which the data in the SDR are to be converted. The spectrum units are specified in the same way as the SDB header variable SPECUNIT. The default value of SPECUNIT is 1, indicating the spectrum is to be live-time normalized.

Discussion

This routine renormalizes the input spectra passed in R_SDR, converting the spectrum into the units specified by SPECUNIT and storing the resultant spectra in the DAT and ERR fields of the corresponding element of R_SDR. The SDB header is also appropriately updated.

SDR$SUB

This procedure performs a channel-by-channel difference of the spectra stored in the two input SDR's, storing the result in an output SDR. The format of this command is:

Calling Sequence

SDR$SUB, r_sdr1, r_sdr2, r_out

Arguments

R_SDR1

The first of two input SDR pointer arrays or subarrays for which the spectral subtraction is to be performed.

R_SDR2

The second of two input SDR pointer arrays or subarrays for which the spectral subtraction is to be performed.

R_OUT

The output SDR pointer array or subarray into which the resulting difference spectra are to be placed.

Keywords

/LTNORM

If present and non-zero, /LTNORM specifies that the input spectra are to be live-time normalized prior to performing the spectral subtraction. The default action is to **not** perform live-time normalization.

SPECUNIT

An integer specifying the spectrum units into which the input spectra are to be converted prior to the spectral subtraction. The spectrum units are specified in the same way as the SDB header variable SPECUNIT. The default value of SPECUNIT is 0, indicating the input spectral units are not to be modified.

MODE

An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 0, indicating that if any indirect fields in the input SDRs are not available, they will be read from the corresponding SDB file and stored in a temporary area.

XSCALE

A real variable specifying a scale factor by which the spectra in R_SDR2 are multiplied prior to the spectral subtraction. The default value of XSCALE is 1.0.

Discussion

This routine performs the spectral subtraction for each element of the input pointer arrays R_SDR1 and R_SDR2, storing the resultant difference spectrum in the output record R_OUT. Any data stored in R_OUT will be overwritten. The SDB header for the output spectrum is constructed from the two input headers as described in Chapter 5. The input spectra are not affected.

If SPECUNIT is not specified, the units are determined from the first spectrum of R_SDR1.

If the units of both of the input spectra are COUNTS, and /LTNORM is not specified, the spectral subtraction is performed directly on the raw count spectra without live-time normalization and the units of the output spectrum will be COUNTS.

The output spectrum and errors are given by:

$$Diff(i) = Spec1(i) - XSCALE \times Spec2(i)$$

$$Err(i) = \sqrt{Err1(i)^2 + (XSCALE \times Err2(i))^2}$$

where i is the spectrum channel number. If /LTNORM is specified, live-time normalization of the spectra is performed prior to the spectral subtraction, if necessary.

In cases where an error condition occurs, all information stored in R_OUT associated with the corresponding spectrum element is deleted and processing continued with the next valid record element.

SDR$SUM

This procedure performs a channel-by-channel sum of **all** spectra stored in the input SDR array, storing the resulting summed spectrum and associated errors into the output SDR. The format of this command is:

Calling Sequence

SDR$SUM, r_sdr, r_out

Arguments

R_SDR

An input SDR pointer array or subarray containing the spectra to sum.

R_OUT

An output SDR pointer or pointer array element into which the result is placed.

Keywords

ERSTAT

A variable to receive an array of status codes, one status code element per SDR element. A status code value of zero indicates that all energy ranges for the element were succesfully included in the sum. Non-zero values must be interpreted as bit picks, see Table 3.2.

Bit	Description
0	Error bit. At least one energy range not included in sum.
1	Low range livetime = 0; not included in sum.
2	Medium range livetime = 0; not included in sum.
3	High range livetime = 0; not included in sum.
Other	An error other than livetime = 0 occurred.

Table 3.2: ERSTAT Keyword Status Codes for SDR$SUM

/LTNORM

If present and non-zero, /LTNORM specifies that the input spectra are to be live-time normalized prior to performing the spectral summation. The default action is **not** to perform live-time normalization.

SPECUNIT

An integer specifying the spectrum units into which the input spectra are to be converted prior to the spectral summation. The spectrum units are specified in the same way as the SDB header variable SPECUNIT. The default value of SPECUNIT is 0, indicating the spectral units are to be taken from the spectrum in R_SDR.

MODE
An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 0, indicating that if any indirect fields in the input SDRs are not available, they will be read from the corresponding SDB file and stored in a temporary area.

/NOWEIGHT
If present and non-zero, /NOWEIGHT specifies that the spectral summing is to be unweighted. This flag is equivalent to setting WEIGHT = 0. The default action is to perform **live-time weighted sums**.

/VERBOSE
If present and non-zero, /VERBOSE specifies that informational messages concerning the type of summing being performed will be displayed. The default is to disable printing of these informational messages.

WEIGHT
An integer value which specifies the weighting method to be used when performing the spectral sum. The supported values are given in Table 3.3.

The default value of WEIGHT = 2, indicating that the sum is weighted by the spectral live-times.

Value	Description
0	Unweighted sum
1	Sum weighted by errors
2	Sum weighted by live-time (DEFAULT)
ELSE	Unweighted sum

Table 3.3: WEIGHT Keyword Values

Discussion

The spectral sum is a channel-by-channel sum of all of the valid spectra passed in R_SDR. The summation is performed for all elements of R_SDR having a spectral type compatible with the first spectrum in the array. If a spectrum in the array is not compatible (*e.g.* different numbers of spectral channels, spectral units which cannot be converted to the required type, etc.), it is not included in the sum. Any data in R_OUT are overwritten. The SDB header for the output spectrum is constructed from the input headers as described in "OSSE Standard Application Processing of Header Quantities", found in Chapter 5. The input spectra are not affected.

Live-time normalization is performed on the input spectra prior to the sum if one or more of the following conditions are met: 1) bit 0 of the keyword SPECUNIT is set, 2) the keyword /LTNORM is specified, or 3) the first spectrum of R_SDR is live-time normalized.

If WEIGHT = 2 (the default value) the spectral summation takes the form of a **live-time weighted mean** of the input spectra and is given by:

$$Sum(i) = \frac{1}{LiveTime_{Rsum}} \sum_{j=1}^{N_{sum}} Spec_j(i) \times LiveTime_{R_i}$$

$$Error(i) = \frac{1}{LiveTime_{Rsum}} \sqrt{\sum_{j=1}^{N_{sum}} (Error_j(i) \times LiveTime_{R_i})^2}$$

where $LiveTime_{R_i}$ is the spectra live-time for the associated spectral range and $LiveTime_{Rsum}$ is the sum of the live-times for the valid spectra for the associated spectral range.

If live-time normalization is **not** set and WEIGHT = 0 or /NOWEIGHT is specified, the spectral summation takes the form of an **arithmetic sum** of the input spectra and is given by:

$$Sum(i) = \sum_{j=1}^{N_{sum}} Spec_j(i)$$

$$Error(i) = \sqrt{\sum_{j=1}^{N_{sum}} Error_j^2(i)}$$

where i is the channel number, j is the spectrum number, and N_{sum} is the number of valid input spectra. Any live-time normalized spectra will be excluded from the spectral summation.

If live-time normalization **is** set and WEIGHT = 0 or /NOWEIGHT is specified, the spectral summation takes the form of an **unweighted mean** of the input spectra and is given by:

$$Sum(i) = \frac{1}{N_{sum}} \sum_{j=1}^{N_{sum}} Spec_j(i)$$

$$Error(i) = \frac{1}{N_{sum}} \sqrt{\sum_{j=1}^{N_{sum}} Error_j^2(i)}$$

The units of the output spectrum will be SPECUNIT, if passed, or that of the first spectrum to be summed.

If WEIGHT = 1, the spectral summation takes the form of a **weighted mean** of the input spectra and is given by:

$$Sum(i) = \frac{\sum_{j=1}^{N_{sum}} \frac{Spec_j(i)}{Error_j^2(i)}}{\sum_{j=1}^{N_{sum}} \frac{1}{Error_j^2(i)}}$$

$$Error(i) = \sqrt{\frac{1}{\sum_{j=1}^{N_{sum}} \frac{1}{Error_j^2(i)}}}$$

The units of the output spectrum will be SPECUNIT, if passed, or that of the first spectrum to be summed.

Warning/Error Messages:

Error Message	Description
Invalid Array Size N.	$Arraysize \leq 0$
... for record # N.	Problem with Spectrum
No Live Time Weight Performed.	If weighted and specunit is counts.
Error initializing first spectrum...	
Error performing sum, element # N.	
Not all spectra summed, N invalid records.	
Inconsistent spectral types...	
Sum is weighted by errors.	
Sum is unweighted sum.	
Sum is weighted by live_time	

Table 3.4: SDR$SUM Error Messages

SDR$UPDHDR

This procedure will update the SDB header stored in an SDR for the specific type of operation to be performed. The command format is:

Calling Sequence

SDR$UPDHDR, r_in, r_out, optype

Arguments

R_IN

An input SDR pointer, or pointer array element, to be merged into the output SDR R_OUT.

R_OUT

An output SDR pointer, or pointer array array element, into which the result is placed.

OPTYPE

An optional character string specifying the type of update operation to perform. Possible OPTYPE values are:

'PDA' – specifies that the SDB header is to be updated using the PDA operation definition. This is the default operation.

'BEST' – specifies that the SDB header is to be updated using the BEST operation definition.

'WRITE' – specifies that the SDB header is to be updated using the WRITE operation definition.

'SUB' – specifies that the SDB header is to be updated using the SUB operation definition.

'SUM' – specifies that the SDB header is to be updated using the SUM operation definition.

Discussion

This routine will update the output SDR's header using information from both the input and output headers. The input header is not affected; however, the output header will be overwritten. The output header is assumed to have been initialized on the first call to SDR$UPDHDR.

The specific updating operation performed on the input and output headers depends on the keywords provided. Only one update operation type may be specified. If OPTYPE is not specified, the headers are updated according to the PDA operation definition.

The specific update operations performed by SDR$UPDHDR are defined in the "OSSE Standard Application Processing of Header Quantities", found in Chapter 5.

Note that, in some cases, multiple calls for arrays of headers are required.

SDR$VIEW

This procedure provides a method for displaying the contents of the HDR and DAT fields from an SDR. The format of this command is:

Calling Sequence

SDR$VIEW, r_sdr

Arguments

R_SDR
An SDR pointer, or single element of a pointer array, for which the contents listing is to be performed.

Discussion

The format of the display is similar to that used to display a record from an SDB file (SDB$LOOK).

Related Routines

See also the related routines LISTREC and SDB$LOOK.

SDR_CHANGE_TARG

This routine converts header and any CRCS in supplied SDRs from current target to input target specification. The format of this command is:

Calling Sequence

SDR_CHANGE_TARG, sdr, targ

[This procedure is written in IDL]

Arguments

SDR
An input/output array of one or more SDR records containing header records.

TARGDEF
An input parameter of either target id, target structure or string specifying target name. If target id, it will be used to look up target in catalog. If target structure, it must have either (ra, dec) fields or (l2,b2) fields, and name field. In some applications, addition of SRCID field is also desired, although if not specified, it will be assigned a SRCID outside the range of the planning catalog. If string, it is used as name for lookup in the catalog.

Keywords

ERSTAT
Status of successful completion (0 - good, 1 - bad).

Discussion

Computes pointing relative to target defined by TARG and replace associated information in SDR.

SDR_COR_MATRIX

This routine creates and returns a correlation matrix built from the covariance data stored in the IGR$COVAR auxiliary fileds in the input sdr array. The format of this command is:

Calling Sequence

SDR_COR_MATRIX, sdr, matrix

[This procedure is written in IDL]

Arguments

SDR
An input array of one or more SDR records containing covariance information.

MATRIX
An output IDL array containing the resultant correlation matrix. This is a two-dimensional array of fltarr(NSDR,NSDR).

Keywords

ERSTAT
A output array containing the SDR_COR_MATRIX status per sdr. Array elements are non-zero if the IGR$COVAR for an associated SDR record was not available.

SDR_ESCALE

This procedure adds edges and widths to SDRs using gain information in header or keyword-supplied edges and widths. The format of this command is:

Calling Sequence

SDR_ESCALE, r_sdr

Arguments

R_SDR
Array of one or more sdr records which have headers.

Keywords

EDGES
An input float array which specifies the lower edge (in MeV) of the output channels. EDGES overrides gain computation and must have the same dimensions as data from the sdr.

WIDTHS
An input float array which specifies the width (in MeV) of the the output channels. Both EDGES and WITHS should be specified to override the default gain computation.

GRATIO
Input specification of gain ratio. If set, gain ratio is used to calculate edges and widths, overriding EDGE and WIDTHS keyword arguments.

ERSTAT
Completion status, non-zero if error(s) occurred.

VERBOSE
Input switch indicating that informational messages should be printed.

SDR_HDR_COMPARE

This procedure compares the SDB header fields stored in two SDRs, displaying a list of the differences. The format of this command is:

Calling Sequence

SDR_HDR_COMPARE, r_sdr1, r_sdr2

[This procedure is written in IDL]

Arguments

R_SDR1
An input SDR pointer, or pointer array element, for which the SDB header comparison is to be performed.

R_SDR2
An input SDR pointer, or pointer array element, for which the SDB header comparison is to be performed.

SDR_LOAD

This procedure creates and loads an array of SDRs from an SDB file. Optionally, selections can be made on the contents of the file to be loaded. The default is header, data, errors, edges, and widths from all records in the file. In addition, data can be optionally summed into summary SDRs.

Calling Sequence

SDR_LOAD, filename, sdr [,outsdr]

[This procedure is written in IDL]

Arguments

FILENAME

A character string specifying the name of the input SDB file. The default directory is the current directory, the default file extension is .SDB. The file must exist.

SDR

An array of SDR records created from the information in the input SDB file and selection criteria. If the SDR array exists on entry, it is deleted and re-created with the information in the file.

OUTSDR

An optional parameter for returning the individual SDRs from the file when the SUM keyword is set to return sums in the SDR argument. If SUM is not specified, this argument is not used.

Keywords

AUXA

This keyword specifies the aux types to be extracted from the SDB records in the SDB file. By default, hdr, dat, err, edg, wid, are read. AUXA can be specified as a string of aux types separated by commas - eg. 'dat,err,edg,wid,crc,note' or as a two element long array specifying "bit-picks". AUXA overrides the default and requires default aux types to be specifies in AUXA if desired.

AUX1FLD

A longword which specifies the AUX1 bits. The default is equivalent to hdr, dat, err, edg, and wid.

AUX2FLD

A longword which specifies the AUX2 bits. The default is zero.

/BYCFG
Input switch which if present and nonzero, specifies that summing should be performed for each unique detector configuration. Default is single sum of available data.

/BYDAY
Input switch which if present and nonzero, specifies that summing should be performed for each (approximate) daily interval. Default is single sum of available data.

/BYDET
Input switch which if present and nonzero specifies that summing should be performed by detector. By default, all sdrs are summed into a single sum. The keyword SUM must be set along with BYDET to create individual sdrs for each detector.

/BYDSA
Input switch which if present and nonzero, specifies that summing should be perfomed by detector step angle or source aspect. By default, all sdrs are summed into a single sum. The keyword SUM must be set along with BYDSA to create individual sdrs for each step angle. BYSCAN overrides BYDSA and a warning is issued. The combination of BYDSA and BYDET will create a #dsa by #det element sdr array.

/BYSCAN
Input switch which if present and nonzero, specifies that summing should be perfomed by target scan angle (HDR.TARGSCAN) or source aspect. By default, all sdrs are summed into a single sum. The keyword SUM must be set along with BYSCAN to create individual sdrs for each scan angle. BYSCAN overrides BYDSA and a warning is issued. The combination of BYSCAN and BYDET will create a #scan by #det element sdr array. NOTE: TARGSCANS are rounded to 0.2 degrees to find unique ranges.

DETID
This keyword specifies which detectors to extract from the the SDB file. Its format is the standard "bit-pick" specification for 1 or more detectors. The default is to select all detector data from the SDB file.

ELEMENTS
An input array of indices of SDB records to be read from the requested SDB file. By default, all elements are read. Simultaneous usage of the keywords, TAGS and ELEMENTS is not supported.

ERSTAT
A output array containing the SDB$READ status. Array elements are non-zero if the data requested for an associated SDR record was not available.

/MATRIX
Input switch, which if present and non-zero, causes summing of response matrices if found in input sdrs. Live time weighted rmtrx is stored in output sdrs.

NOT_ELE
An input array of indices of SDB records to exclude. By default, all elements are read.

NOT_TAGS
An input array of one or more tagwords of SDB records to be excluded. By default, all tags are read.

PLUS_AUX
This keyword specifies which additional aux types should be loaded. The standard dat, err, edg, wid will be loaded, plus those specified with this keyword.

/RELSPEC
Input switch which if present and nonzero, specifies that RELSPEC records should be added to input SDR. The RELSPEC records document the file name and tagword of the file and SDR containing the summed data.

START_TIME
Input specifying start time limit for selection. Records with start times greater or equal to this time are selected. Time is specified as tjd and fraction in one of many forms. eg. as string - 'yy/ddd hh:mm:ss', as real - yyddd.fff. See routine TJD_FROM_ASCII. Default TJD_START is 0, ie accept all records.

STOP_TIME
Input specifying stop time limit for selection. Records with stop times less than or equal to this time are selected. Time is specified as tjd and fraction in one of many forms. eg. as string - 'yy/ddd hh:mm:ss', as real - yyddd.fff. See routine TJD_FROM_ASCII. Default TJD_STOP is 1e10, ie accept all records.

/SUM
An input logical which if present and nonzero indicates that the sum of the individual records should be performed and returned. By default, a single sum is generated. Keyword BYDET changes this default action. If sdrs exists upon entry, they are delete and output sdrs for the sums are created.

TAGS
An input array of one or more tagwords of SDB records to be read from the requested SDB file. By default, all tags are read.

/UNIQUE
Input switch, which if present and non-zero, indicates that unique logical names should be created for each input sdb filename. The default is to abort if duplicate filenames are found, thus /UNIQUE permits using the same filename in different directories.

WEIGHT
An integer value which specifies the weighting method to be used when performing the spectral sum. The supported values are given in Table 3.3. Default is 2 (LTNORM).

Discussion

```
EXAMPLE:

  #1 get sdrs from file for dets 1 and 2 and sum into individual sums

     igr>  sdr_load, '$prod:p931230000_dif.sdb', sdr_sums, $
     igr>            /sum,  /bydet,  detid = 3

  #2 get sdrs from file with footnotes, crc, and psdeff data

     igr> sdr_load, filename, sdr, auxa = 'dat,err,edg,wid,note,crc,psdeff'
```

Related Routines

Also see the related function SDR_SUM.

SDR_REBIN

This procedure provides a method for rebinning data stored in an SDR into new energy bins. The format of this command is:

Calling Sequence

SDR_REBIN, r_sdr, newedg, newwid

[This procedure is written in IDL]

Arguments

R_SDR
An input SDR pointer array or subarray containing the data to be rebinned. The SDRs must have a current energy scale identified so the initial bins can be determined.

NEWEDG
An input array of lower edges energies for the new binning.

NEWWID
An input array of energy widths for the new binning.

Keywords

ERSTAT
Output array, one element per input SDR, which indicates error in processing request. Nonzero is error, zero is success.

INMAP
Input array which provides an alternate method for mapping input channels into output channels. One element in INMAP for every channel in the input SDR. Content of the element is the output channel # (0...Nout-1). Content of less than zero indicates that the channel is not mapped to any output channel. INMAP usage requires /NOFRACTIONS specification as well. If INMAP is provided, parameters NEWEDG and NEWWID are ignored on input, but return the edges and widths of the first SDR that are identified by the INMAP.

/NATURAL
If present and non-zero, /NATURAL turns off the warning message that the input edge definition is monotonic increasing.

/NOFRACTIONS
If present and non-zero, /NOFRACTIONS specifies that the initial bins are not to be fractionated when converting to the new binning. In this case, the new edges and

new widths are slightly modified to coincide with the nearest values from the initial bins.

OUTMAP

Returned array of the form of INMAP which indicates that the binning was performed. Valid only in conjunction with /NOFRACTIONS.

RANGE_SWITCH

An input 2-element array which defines the energies at which binning changes from LOW to MEDIUM and from MEDIUM to HIGH spectral ranges. The default is to take the highest channel of each range (*e.g.*, use all of the LOW range, then the remaining portion of the MEDIUM range).

Discussion

This procedure rebins the data stored in R_SDR into the new energy bins specified by NEWEDG and NEWWID. The input SDRs must have current energy scale identified. The rebinning is performed "in place"; that is, the original SDR contents are over-written. The SDR header variables are updated to reflect data arrays. The units of the returned data are MeV^{-1}. The appropriate EDG and WID AUX fields are also created.

SDR_REBIN creates an sdr data array which appears as one energy range - the low range; ie. NCHANL and LTL describe the complete data array. NCHANM and NCHANH are zero; LTL, LTM and LTH remain what they were, but since only NCHANL is nonzero, livetime computations with the resultant spectrum only use LTL.

For this reason, SDR_REBIN should generally be given live-time normalized spectra as input, or differences in LTL,LTM,LTH will not be correctly handled in converting the counts into rates.

Related Routines

Also see the related functions RESAMPLE and MERGECHAN.

SDR_SETDATE

This routine updates the field HDR.CREDATE in the SDB header of each input SDR. Default is the current date-time. The format of this command is:

Calling Sequence

SDR_SETDATE, r_sdr

Arguments

R_SDR
Input SDR (or SDR array) for which field HDR.CREDATE will be updated with current date-time.

Keywords

DATE
Input string specifying the date to be placed in the SDB header field HDR.CREDATE of each SDR. The format is "dd-mmm-yyyy 00:00:00.0".

Discussion

This routine updates the SDB header field HDR.CREDATE of each input SDR.

SDR_SPECUNIT

This routine changes the units of the data in the input sdr array to the specified units. It supports time and energy normalization (and unnormalization). The format of this command is:

Calling Sequence

SDR_SPECUNIT, sdr

[This procedure is written in IDL]

Arguments

SDR

An input/output array of one or more SDR records containing at least headers and data. Depending on specified units, may also require access to errors,edges and widths. Required aux types are loaded into the SDRs if necessary.

Keywords

MEV

An input logical which if present and nonzero indicates that the spectrum should be changed to "per MeV". Requires access to errors, edges and widths (will calculate them if it can). If present and zero, the energy normalization of the sdr is removed if necessary.

NOMEV

An input logical which if present and nonzero indicates that energy normalization should be removed. This is the same as specifying MEV=0.

NOSEC

An input logical which if present and nonzero indicates that time normalization should be removed. This is the same as specifying SEC=0.

SEC

An input logical which if present and nonzero indicates that the spectrum should be livetime normalized, if necessary. Requires access to errors (will calculate them if it can). If present and zero, the time normalization of the sdr is removed if necessary.

Discussion

NOTE: A call to this routine with no keywords does nothing.

SDR_SUM

This procedure sums an array of IGORE SDRs into an output SDR. The format of this command is:

Calling Sequence

SDR_SUM, in_sdr, out_sdr

Arguments

IN_SDR
Array of one or more Igore SDR records which have HDR, DAT, and ERR fields. The SDB header must be loaded, this routine will load others as needed.

OUT_SDR
A single SDR in which to store the sum of the input SDRs. OUT_SDR must exist on call. It is returned with HDR, DATA, ERR, EDG, and WID fields populated.

Keywords

AUX
Input string specifying list AUX data types required to be in the input SDRs. Warning messages are printed if these data types are not in input SDRs.

NOTE: Currently supports only CRCs.

/BYCFG
Input switch which if present and nonzero, specifies that summing should be performed for each unique detector configuration. Default is single sum of available data.

/BYDAY
Input switch which if present and nonzero, specifies that summing should be performed for each (approximate) daily interval. Default is single sum of available data.

/BYDET
Input switch which if present and nonzero, specifies that summing should be performed by detector. Default is a single sum of available data.

/BYDSA
Input switch which if present and nonzero, specifies that summing should be perfomed by detector step angle or source aspect. By default, all sdrs are summed into a single sum. The keyword SUM must be set along with BYDSA to create individual

sdrs for each step angle. BYSCAN overrides BYDSA and a warning is issued. The combination of BYDSA and BYDET will create a #dsa by #det element sdr array.

/BYSCAN
Input logical which if present and nonzero, specifies that summing should be perfomed by target scan angle (HDR.TARGSCAN) or source aspect. By default, all sdrs are summed into a single sum. The keyword SUM must be set along with BYSCAN to create individual sdrs for each scan angle. BYSCAN overrides BYDSA and a warning is issued. The combination of BYSCAN and BYDET will create a #scan by #det element sdr array. NOTE: TARGSCANS are rounded to 0.2 degrees to find unique ranges.

COR_MATRX
Output IDL structure containing the correlation matrix.

CRC_SUM
Output IDL structure containing the resultant CRCs from the sum.

EDGES
Input float array which specifies the lower edges (MeV) of the output channels. Overrides default gain computation. Must have same dimensions as DAT field in SDR.

/MATRIX
Input switch, which if present and nonzero, causes summing of response matrices in input sdrs to be performed and stored in output sdrs.

/RELSPEC
Input switch which if present and nonzero, specifies that RELSPEC records should be added to input SDR. The RELSPEC records document the file name and tagword of the file and SDR containing the summed data.

/VERBOSE
Logical switch specifying that information messages should be displayed. Default is not to display information messages.

WEIGHT
An integer value which specifies the weighting method to be used when performing the spectral sum. The supported values are given in Table 3.3. Default is 2 (LTNORM).

WIDTHS
Input float array which specifies the widths (MeV) of the output channels. Both edges and widths should be specified to override default gain computation.

Discussion

SDR_SUM performs a live-time weighted sum of an array of SDRs into an output SDR.

The summing assumes that the input array of SDRs have HDR, DAT, and ERR fields populated. SDR$SUM is called to perform the summing of data and correct update of header information.

OUT_SDR should be created before calling SDR_SUM.

NOTE: Edges and widths returned in OUT_SDR are obtained via one of several alternatives. If edges and widths are supplied via the keywords EDGES and WIDTHS and have the correct dimensions, the supplied edges and widths are used. Otherwise, if edges and widths are found in the IN_SDR or in the corresponding SDB file, the average edges and widths are computed. If not, and the input data has 544 channels (no rebinning), edges and widths are computed by SDR_ESCALE using the SDB header RGAIN values in IN_SDR, otherwise no edges and widths are returned.

NOTE: If input IN_SDR array contains data from multiple detectors, they all get summed into a single SDR_OUT with channel edges and widths averaged - garbage in - garbage out, unless the /BYDET keyword is used to sum by detector.

Related Routines.

See the related routine SDR$SUM.

SELECT_FILES

This function selects a set of files using "wildcard" filename input specifications and assigns logical names for each file which translate to the full file specification. The logical name is formulated from the "Name" portion of the full file specification. In interactive mode, this function provides a listing of files satisfying the "wildcard" filename input specification allows the user to select files from the list.

Optional keywords provide tracking of logical names and associated full file specifications.

The format of this command is:

Calling Sequence

lognames = SELECT_FILES(file_spec)

[This procedure is written in IDL]

Arguments

FILE_SPEC

A character string (or string array) containing the user's file specification. VMS wildcard file specifications are supported. If an array of file specifications, each selection being processed in turn.

lognames

A character string (or string array) of logical names assigned to the selected files in the order of selection.

Keywords

/ALL

If present and non-zero, /ALL indicates that **all** files satisfying the user file specification will be returned without prompting the user for further file selection.

LOGMAP

The name of an IDL variable which, if present, upon return contains an IDL record of structure type FILE_LOGS containing the pairing of logical names and full file specifications.

FULLNM

The name of an IDL variable which, if present, upon return contains an array of full file specifications which correspond to the returned lognames.

/NOWARN

If present and non-zero, /NOWARN turns off the informational warning message relating to the reassignment of an existing logical. The default action is to print a warning message when an existing logical is reassigned.

MAXLOG

An input integer specifying the maximum length for the logical name assigned to the file specification. The default is 20 characters.

UNIQUE

Input switch which if present and nonzero, specifies that an alphabetic charater be added as a suffix to the fileroot, if required, to create a unique logical name.

Discussion

This procedure assigns logicals to the files passing the file specification FILE_SPEC, returning the logicals in the returned character string LOG_MAP. If multiple files satisfy the file specification and the keyword /ALL is not specified, a list of filenames which match the input file specification is generated and the user is prompted to enter by number the file(s) from the list for which logicals should be assigned. Input numbers can be separated by spaces or commas, but must be entered on a single line.

If only one file meets the wildcard specification, or if the keyword /ALL is provided, no list is generated and no prompting is performed.

The process logical names are obtained from the file root and will be assigned or reassigned. An informational message indicates the reassignment of an existing logical name, unless the keyword /NOWARN is specified.

Related Routines

See the related routine RESTORE_LOGS for re-assignment of logicals using IDL records of structure type FILE_LOGS, which may be returned by SELECT_FILES using the keyword LOGMAP.

SETUP_REC

This procedure reads data from an ASCII file into fields of an IGORE record. The format of this command is:

Calling Sequence

SETUP_REC, r_pointer, input_file

[This procedure is written in IDL]

Arguments

R_POINTER
A single IGORE record pointer of any structure type.

INPUT_FILE
A character string specifying the file containing the information to be stored in the IGORE record.

Discussion

This procedure is used to set the value of a number of fields in an IGORE record without directly and repeatedly calling PUTFLD. The values to be stored are contained in an ASCII input file.

The input file should have one field per line, of the form:

FIELDNAME = value

where FIELDNAME is the name of the field in the IGORE structure. For example,

EPHEM_PHI0= 0.52720433

for numeric variables and

PDF_FILE='dka500:[pulsar]p911400000.pdf'

for strings. (Note that strings may be delimited by single quotes, double quotes, or angle brackets (*e.g.*, pdf_file=⟨p911400000.pdf⟩).) Specification of field names is not case sensitive. Input field names are checked for validity. Spaces are ignored (except inside character strings), but don't use tabs. Procedure does not currently handle arrays of records or storing arrays into fields.

SHOWFLD

This procedure produces a listing of the fields contained in a specified IGORE structure. The format of this command is:

Calling Sequence

SHOWFLD [, structure_name]

Arguments

STRUCTURE_NAME
An optional character string specifying the name of the IGORE structure for which the listing is to be performed. If this argument is not provided, a listing of all of the known IGORE structures is displayed and the user is prompted to select the structure to be displayed.

Related Routines

See the related routines FLDNAME, which returns a string array specifying all of the field names for a given IGORE structure type, and SHOWSTRUC, which lists all of the IGORE structures.

SHOWIGORE

This procedure produces a detailed listing of the current IGORE session environment parameters. The format of this command is:

Calling Sequence

SHOWIGORE

[This procedure is written in IDL.]

Arguments

NONE

Keywords

/BRIEF
If present and non-zero, /BRIEF indicates a summary listing of IGORE environment parameters is to be displayed. equivalent to specifying /PARAMETERS, /STRUCTURES, and /RECORDS.

/PARAMETERS
If present and non-zero, /PARAMETERS indicates that the IGORE virtual memory parameters are to be displayed.

/STRUCTURES
If present and non-zero, /STRUCTURES indicates that the currently defined IGORE structures are to be listed.

/RECORDS
If present and non-zero, /RECORDS indicates that the currently defined IGORE records are to be listed and the corresponding virtual memory allocations are to be displayed.

TABLE
A character string indicating the IGORE virtual memory management table for which information is to be displayed. Valid table types are: "RECORD", "STRUCTURE", "INDIRECT" and "TEMPORARY".

Discussion

This routine provides the ability to display various information about the current IGORE session. If no keywords are provided, a complete listing is produced which displays the IGORE parameters, structures, records, and all four memory management tables. (the same as if the keywords /PARAMETERS, /STRUCTURES, /RECORDS, and TABLE were provided).

Related Routines

See the related routines SHOWSTRUC and SHOWREC.

SHOWREC

This procedure provides information about IGORE records defined for the current session. The format of this command is:

Calling Sequence

SHOWREC [, r_record]

Arguments

R_RECORD
An optional IDL variable containing the record pointers to the IGORE record for which the record statistics are to be generated. If not passed, statistics on all IGORE records will be listed.

Discussion

This routine lists the currently defined IGORE records, including their sizes, types, record length and the VM allocated for direct and indirect fields. If the optional parameter R_RECORD is provided, only the record statistics for the specified record will be displayed.

SHOWSTRUC

This procedure produces a listing of the IGORE structures for the current session. The format of this command is:

Calling Sequence

SHOWSTRUC

Arguments

NONE

Discussion

This routine lists the currently defined IGORE structures, including the number of fields and the total direct memory requirements.

Related Routines

See also the related routine SHOWFLD.

SHOW_CMDSTATE

This procedure displays the command state information from the OSSE Command State Database. The format of this command is:

Calling Sequence

SHOW_CMDSTATE

[This procedure is written in IDL]

Arguments

NONE (one of the keywords, STATES or TJD_START, is required).

Keywords

STATES
An input string, or string array, specifying the command state name(s) to be displayed. The command state names are character strings of the form "yydddnnn" (*e.g.*, '91136001') indicating the year day and version number of the command state. Only one of the keywords STATES or TJD_START should be specified.

TJD_START
An input variable specifying the start time of the interval for which command state information is requested. Only one of the keywords STATES or TJD_START should be specified. The keyword TJD_START may take one of the following forms:

String –	A string date specification of the form "yy/ddd hh:mm:ss". Any parts of the date not specified will be interpreted as zero.
Real or Double –	The date specification is assumed to be in Truncated Julian days.
Long Integer –	The date specification is assumed to be in PB5 format.

NDAYS
An input variable specifying the length of the interval for which the command state information is to be displayed. This keyword is only valid when used in conjunction with the keyword TJD_START. The default value of NDAYS is 14 days (two weeks).

OUTFILE
An input character string specifying the name of the output file to which the command state information is to be written. If provided, the output file is opened and

closed within the procedure. If not provided, the command state information is written to the screen.

TABLES

An input character string, or string array, specifying the name(s) of the command state tables for which information is requested. See Table 3.5 for a list of valid table names. The specified table names may be abbreviated, in which case all tables satisfying the given abbreviation will be selected (*e.g.*, the table specification "CFG" would select tables "CFG1_TBL", "CFG2_TBL", "CFG3_TBL" and "CFG4_TBL").

DETAILED

Input switch which, if present and non-zero, specifies that the contents of the specified tables are summarized. The default action is to list the table filename only. This keyword operates in conjunction with the TABLES keyword.

DBASE

The name of an IDL variable which, if present, upon return contains structured array of type STATE_VEC which describes the selected state(s). The structure STATE_VEC includes the state name, observation id, encoded times, and a comment string.

Table 3.5: Command State Table Names

PE_TBL	FLTPATCH_TBL	RAMBO_TBL	PATCH_TBL
AGC_TBL	MSR_TBL	VECTOR_TBL	CO60_TBL
CAL_TBL	MDPPRI_TBL	MDPSEC_TBL	MDPSOL_TBL
MDPTRN_TBL	RIUCMD_TBL	PSR_TBL	RPHA1_TBL
RPHA2_TBL	FORMATA_TBL	FORMATB_TBL	BURST_TBL
MACRO_TBL	SPM_TBL	PST_TBL	CFG1_TBL
CFG2_TBL	CFG3_TBL	CFG4_TBL	DTW_TBL
MDC_TBL	UVT_TBL	EAT_TBL	MXT_TBL
PSR_SEC_TBL	PST_SEC_TBL	PSR_SOL_TBL	PST_SOL_TBL
PSR_TRN_TBL	PST_TRN_TBL	USR_PATCH_TBL	CHNSM_PRI_TBL
CHNSM_SEC_TBL	CHNSM_SOL_TBL	CHNSM_TRN_TBL	

Discussion

This procedure is an interface into the OSSE Command State Database and provides a method of displaying the command state information for the specified state (using STATES) or time period (using TJD_START). The specific information to be displayed can be specified using the keyword TABLES. More detailed information can be displayed using the keyword DETAILED.

SHOW_CONFIG

This procedure provides a graphic display of the OSSE collimator pointing configuration based on the specified spacecraft attitude and OSSE target list. The format of this command is:

Calling Sequence

SHOW_CONFIG, attitude, target_cat

[This procedure is written in IDL]

Arguments

ATTITUDE
An input structure of type ATTSTRUC which defines the spacecraft attitude and observation interval for which the display is to be performed.

TARGET_CAT
An input structure scalar or array of type PLANCAT containing information about the sources for which the display is to be performed.

Keywords

/GALACTIC
If present and non-zero, /GALACTIC specifies that the display is to be performed in galactic coordinates. The default is to perform the display in J2000 celestial coordinates.

CAT
An input structure scalar or array of type PLANCAT containing information about other sources which are to be included in the display.

PROJ
An input projection specification used to override the default Mollweide projection. See the IDL User's Library, MAP_SET, for the supported codes.

RANGE
A 2-element array specifying the scan and trans-scan angle range to be displayed on either side of the target position. The default values are 20 degrees.

TITLE
An input character string specifying the title to be displayed on the plot. The default is to display the viewing period number and the target name.

/COLORS
If present and non-zero, /COLORS specifies that the display is to be performed in color. The default is to display the plot in monochrome.

OFFSETS
An input scalar or array of scan offset angles for which the OSSE field-of-view is to be displayed. The angles are specified in degrees relative to the calculated target scan angle. The default displays only a single field of view at the target position (offset scan angle = 0 degrees).

MINPRIO
An input scalar integer specifying the minimum priority of targets in the target catalog CAT that are to be displayed. This keyword is used in conjunction with the keyword CAT. The default value is 3 (*e.g.*, display only targets with priorities greater than or equal to 3).

LABEL
An input scalar integer specifying the minimum priority of targets in the target catalog CAT which are labeled. The default is to display no labels.

/FIELDLAB
If present and non-zero, /FIELDLAB specifies that each of the scan offsets specified by OFFSETS are to be labeled with their index in the OFFSETS array. The default is to produce no field labels.

/NOERASE
If present and non-zero, /NOERASE indicates that the screen is not to be erased before displaying the plot. The default is to erase the screen before displaying the plot.

/OPLOT
If present and non-zero, /OPLOT causes the new configuration to be overplotted on the existing display.

FOV_RESP
An input real value (0.0–1.0) specifying the fractional response level for which the instrument field of view is to be plotted. The default value is 0.1.

P0
The name of an IDL variable which, if present, on return contains the coordinates for the center of the map. This parameter is useful when subsequent calls to SHOW_CONFIG are performed with the keyword /OPLOT.

LIMIT
The name of an IDL variable which, if present, on return contains the coordinates of the corners of the map. This parameter is useful when subsequent calls to SHOW_CONFIG are performed with the keyword /OPLOT.

GLINESTYLE
The line style index used to draw the grid of parallels and meridians. The default is dotted.

GLINETHICK
The thickness of the gridlines for the plot. The default is 1.

/ALLSKY
Input logical which, if set, will display the entire sky, ignoring input range and limits.

/NODATA
Input logical, if set, specifies only the grid is drawn.

LONDEL
Input longitude grid delta per map_set.

LATDEL
Input latitude grid delta per map_set.

/SET_LIMIT
Input logical which, if set, indicates that the information in keywords P0 and LIMIT be used to control display, otherwise use auto scaling and ranging .

LIMIT
The name of an IDL variable which, if present, on return contains the coordinates of the corners of the map. This parameter is when subsequent calls to SHOW_WHEN are performed with the keyword /OPLOT.

Discussion

See the routine SC_ATTITUDE for reading the OSSE Planning Catalog to generate a variable of structure type ATTSTRUC for the parameter ATTITUDE and the routine READ_PLANCAT for reading the OSSE Target Database to generate a variable of structure type PLANCAT for the parameter TARGET_CAT.

SHOW_MDP_CONFIG

This procedure provides a graphical display of the OSSE collimator pointing configuration based on the specified motor-drive positioning (MDP) table. The format of this command is:

Calling Sequence

SHOW_MDP_CONFIG, mdpfile, target_cat

[This procedure is written in IDL]

Arguments

MDPFILE
A character string file specifying the MDP table filename to be displayed.

TARGET_CAT
An input structure scalar or array of type PLANCAT containing information about the sources for which the display is to be performed.

Keywords

/GALACTIC
If present and non-zero, /GALACTIC specifies that the display is to be performed in galactic coordinates. The default is to perform the display in J2000 celestial coordinates.

CAT
An input structure scalar or array of type PLANCAT containing information about other sources which are to be included in the display.

DETID
An input integer specifying the detector(s) to display. This parameter is a bit-pick (like the SDB header variable DETID) in which bits 0–3 represent detectors #1–4, respectively. The default value of DETID is 15, indicating that all detectors are to be displayed.

/FULL
If present and non-zero, /FULL indicates that each detector configuration is to be displayed on a full page. The default is for all four detectors to be displayed on a single page.

PROJ
An input projection specification used to override the default Mollweide projection. See the IDL User's Library, MAP_SET, for the supported codes.

RANGE
A 2-element array specifying the scan and trans-scan angle range to be displayed on either side of the target position. The default values are 20 degrees.

TITLE
An input character string specifying the title to be displayed on the plot. The default is to display the viewing period number and the target name.

/COLORS
If present and non-zero, /COLORS specifies that the display is to be performed in color. The default is to display the plot in monochrome.

MINPRIO
An input scalar integer specifying the minimum priority of targets in the target catalog CAT that are to be displayed. This keyword is used in conjunction with the keyword CAT. The default value is 3 (*e.g.*, display only targets with priorities greater than or equal to 3).

LABEL
An input scalar integer specifying the minimum priority of targets in the target catalog CAT which are labeled. The default is to display no labels.

FOV_RESP
An input real value (0.0–1.0) specifying the fractional response level for which the instrument field of view is to be plotted. The default value is 0.1 degree.

P0
The name of an IDL variable which, if present, on return contains the coordinates for the center of the map. This parameter is useful when subsequent calls to SHOW_MDP_CONFIG are performed with the keyword /OPLOT.

GLINESTYLE
The line style index used to draw the grid of parallels and meridians. The default is dotted.

GLINETHICK
The thickness of the gridlines for the plot. The default is 1.

/OPLOT
If present and non-zero, /OPLOT causes the new configuration to be overplotted on the existing display.

LIMIT
The name of an IDL variable which, if present, on return contains the coordinates of the corners of the map. This parameter is useful when subsequent calls to SHOW_CONFIG are performed with the keyword /OPLOT.

POSITION
Input array, usual plotting meaning.

MDPTABLE
Input MDP table structure. If specified, the MDP definition in the structure is used instead of the MDPFILE parameter (ie. the file is not read in.)

/ALLSKY
Input logical which, if set, will display the entire sky, ignoring input range and limits.

LONDEL
Input longitude grid delta per map_set.

LATDEL
Input latitude grid delta per map_set.

/SET_LIMIT
Input logical which, if set, indicates that the information in keywords p0 and limit be used to control display, otherwise use auto scaling and ranging.

Discussion

See the routine SHOW_CMDSTATE for identifying the MDP file for a specified observation period and the routine READ_PLANCAT for reading the OSSE Target Database to generate a variable of structure type PLANCAT for the parameter TARGET_CAT.

SHOW_PERIOD

This procedure provides a graphic display of the OSSE collimator pointing configuration based on the specified spacecraft attitude and OSSE target list. The format of this command is:

Calling Sequence

SHOW_PERIOD, attitude, target_cat

[This procedure is written in IDL]

Arguments

ATTITUDE

An input structure of type ATTSTRUC which defines the spacecraft attitude and observation interval for which the display is to be performed.

TARGET_CAT

An input structure scalar or array of type PLANCAT containing information about the sources for which the display is to be performed.

Arguments

/GALACTIC

If present and non-zero, /GALACTIC specifies that the display is to be performed in galactic coordinates. The default is to perform the display in J2000 celestial coordinates.

PROJ

An input projection specification used to override the default Mollweide projection. See the IDL User's Library, MAP_SET, for the supported codes.

RANGE

A 2-element array specifying the scan and trans-scan angle range to be displayed on either side of the target position. The default values are 20 degrees.

TITLE

An input character string specifying the title to be displayed on the plot. The default is to display the viewing period number and the target name.

/COLORS

If present and non-zero, /COLORS specifies that the display is to be performed in color. The default is to display the plot in monochrome.

OFFSETS

An input scalar or array scan offset angles for which the OSSE field-of-view is to be displayed. The angles are specified in degrees relative to the calculated target scan angle. The default displays only a single field of view at the target position (offset = 0).

Discussion

See the routine SC_ATTITUDE for reading the OSSE Planning Catalog to generate a variable of structure type ATTSTRUC for the parameter ATTITUDE and the routine READ_PLANCAT for reading the OSSE Target Database to generate a variable of structure type PLANCAT for the parameter TARGET_CAT.

SHOW_STATETABLE

This procedure displays the selected strategy table information from the OSSE Command State Database. The format of this command is:

Calling Sequence

SHOW_STATETABLE, tables

[This procedure is written in IDL]

Arguments

TABLES

An input character string, or string array, specifying the name(s) of the command state tables for which information is requested. See Table 3.5 for a list of valid table names. The specified table names may be abbreviated, in which case all tables satisfying the given abbreviation will be selected (*e.g.*, the table specification "CFG" would select tables "CFG1_TBL", "CFG2_TBL", "CFG3_TBL" and "CFG4_TBL").

NOTE: One of the keywords, STATES or TJD_START, is required.

Keywords

STATES

An input string, or string array, specifying the command state name(s) to be displayed. The command state names are character strings of the form "yydddnnn" (*e.g.*, '91136001') indicating the year day and version number of the command state. Only one of the keywords STATES or TJD_START should be specified.

TJD_START

An input variable specifying the start time of the interval for which command state information is requested. Only one of the keywords STATES or TJD_START should be specified. The keyword TJD_START may take one of the following forms:

String – A string date specification of the form "yy/ddd hh:mm:ss". Any parts of the date not specified will be interpreted as zero.

Real or Double – The date specification is assumed to be in Truncated Julian days.

Long Integer – The date specification is assumed to be in PB5 format.

NDAYS

An input variable specifying the length of the interval for which the command state information is to be displayed. This keyword is only valid when used in conjunction with the keyword TJD_START. The default value of NDAYS is 14 days (two weeks).

OUTFILE

An input character string specifying the name of the output file to which the command state information is to be written. If provided, the output file is opened and closed within the procedure. If not provided, the command state information is written to the screen.

DETAILED

Input switch which, if present and non-zero, specifies that the contents of the specified tables are summarized. The default action is to list the table filename only.

TBLFILES

The name of an IDL variable which, if present, upon return contains a string array containing the file names for the command state tables specified by TABLES. The dimensionality of TBLFILES is (NTABLE, NSTATES), where NTABLES is the number of tables and NSTATES is the number of command states.

QUIET

Input switch which, if present and non-zero, disables all informational messages, overriding the specification of OUTFILE.

DBASE

Output IDL structure array of type STATE_2 containing information about the identified command states.

Discussion

This procedure is an interface into the OSSE Command State Database and provides a method of displaying the command state information for the specified state (using STATES) or time period (using TJD_START). The specific information to be displayed can be specified using the argument TABLES. More detailed information can be displayed using the keyword DETAILED.

SHOW_WHEN

This procedure provides graphic display of OSSE Motor Drive Positioning (MDP) strategy table data.

Calling Sequence

SHOW_WHEN, targname, startdate, stopdate

Arguments

TARGNAME
Input string specifying target of interest from target database. All MDP tables active within the period specified period (STARTDATE to STOPDATE) are checked to see if specified target is within MAX_SCAN degrees of the center of a detector FOV. (Ignored if keyword TARGDEF is used.)

STARTDATE
Input string specifying starting date for cmdstate database check in yyddd hh:mm:ss format, eg. '92/100'

STOPDATE
Input string specifying stoping date for cmdstate database check in yyddd hh:mm:ss format, eg. '92/100'

Keywords

MAX_SCAN
Input, number of scan degrees from specified target to MDP defined detector FOVs. Default is 5 degrees. At least 1 MDP FOV must be within this scan angle from target.

MAX_XSCAN
Input, number of trans-scan degrees from specified target to MDP defined detector FOVs. Default is 10 degrees.

GALACTIC
If specified, plot in galactic coordinates rather than default J2000 celestial coords.

DETID
Input "bit-pick" indicating which detector configurations should be displayed. Default is all (15).

FULL
Input if set, each detector configuration is displayed on a full page. Default is all

four detectors on a single display page.

PROJ
Projection specification to override the default Mollweide projection. See IDL User-slibrary, MAP_SET, for codes.

RANGE
Input specification of scan angle range and trans-scan angle range around target to be displayed. This is a two element array, [scan offset angle, trans-scan offset angle]. Default values are 20 degrees for each; ie display $+/-20$ deg in scan and $+/-20$ degs in trans-scan.

TITLE
Input string which replaces default title on display.

/COLORS
Input, if set, displays in color, else monochrome.

MINPRIO
Input, minimum priority of targets in target catalog to be displayed. Default setting is 3, ie. display priority 3 and 4 targets.

LABEL
Input, specifies minimum priority of targets which are to be labeled. Default is no labels.

FOV_RESP
For field of view displays, this specifies the response level (0...1) for the FOV contour display. If not specified, the default is 0.1 (10 percent).

INTJD
Show time labels in TJD rather than yyddd for JEG

CONFIG
If set, list states, without generating plots

TARGDEF
Input IDL structure which specifies a target other than those found in the Planning Target Catalog. This structure is generally of the form of the target catalog structure, but only REQUIRES a coordinate pair (either RA,Dec or L2,B2) See Table 3.6 for fields of interest.

VIEWPERIOD
Input specification of viewing period to examine. This overrides any specification of start and stop dates.

TABLETYPE
Input string array of one or more positioning table type names. Valid names are "PRI", "SEC", "SOL" and "TRANS". Only specified positioning table types will

be examined. Default is to examine all four types.

OPLOT
Input which if set, causes new configuration to be overplotted on existing display.

/ALLSKY
Input logical which, if set, will display the entire sky, ignoring input range and limits.

P0
The name of an IDL variable which, if present, on return contains the coordinates for the center of the map. This parameter is useful when subsequent calls to SHOW_WHEN are performed with the keyword /OPLOT.

/SET_LIMIT
Input logical which, if set, indicates that the information in keywords igrpramp0 and igrparamlimit be used to control display, otherwise use auto scaling and ranging .

LIMIT
The name of an IDL variable which, if present, on return contains the coordinates of the corners of the map. This parameter is useful when subsequent calls to SHOW_WHEN are performed with the keyword /OPLOT.

LONDEL
Input longitude grid delta per map_set.

LATDEL
Input latitude grid delta per map_set.

GLINESTYLE
The line style index used to draw the grid of parallels and meridians. The default is dotted.

GLINETHICK
The thickness of the gridlines for the plot. The default is 1.

/NOPRI
An input logical flags, if present and nonzero will prevent analysis of the associated pointing table type, "PRI" when validating viewing configurations. See TABLETYPE keyword.

/NOSEC
An input logical flag, if present and nonzero will prevent analysis of the associated pointing table type, "SEC" when validating viewing configurations. See TABLETYPE keyword.

/NOSOL
An input logical flag, if set (nonzero) will prevent analysis of the associated pointing table type, "SOL" when validating viewing configurations. See TABLETYPE keyword.

/NOTRANS

An input logical flag, if present and nonzero will prevent analysis of the associated pointing table type "TRANS" when validating viewing configurations. See TABLE-TYPE keyword.

Field Name	Description
NAME	Character string name (12 chars max used)
RA	Float, J2000 RA in degrees
DEC	Float, J2000 Dec in degrees
L2	Float, Galactic longitude in degrees
B2	Float, Galactic latitude in degrees
SRCID	Integer, catalog ID
PRIORITY	Integer, Source Priority

Table 3.6: TARGDEF Field Names

Discussion

This procedure provides graphic display of OSSE Motor Drive Positioning (MDP) strategy table, Including GRO attitude and OSSE pointing offsets based on information in viewing plan, command state database and specified date range.

SOLAR_RADEC

This procedure calculates the solar position in celestial coordinates for the specified epoch. The format of this command is:

Calling Sequence

SOLAR_RADEC, tjd, sol_ra, sol_dec

[This procedure is written in IDL]

Arguments

TJD
The Truncated Julian day, including fractional day, for which the solar position is to be determined.

SOL_RA
The Right Ascension of the sun, in radians, for the epoch specified by TJD.

SOL_DEC
The Declination of the sun, in radians, for the epoch specified by TJD.

STRUCEXIST

This function returns a logical flag indicating whether the specified IGORE structure exists. The format of this command is:

Calling Sequence

flag = STRUCEXIST(struc_name)

Arguments

STRUC_NAME
A character string specifying the name of the IGORE structure which is to be searched for in the IGORE structure table.

Discussion

This routine determines whether an IGORE structure exists having the name specified in the argument STRUC_NAME. The returned function value FLAG is a byte variable indicating whether the an IGORE structure exists having the specified name. If a structure exists having the specified name, the returned value of FLAG will be 255 (true); otherwise, the returned value will be 0 (false).

Related Routines

See the related routine SHOWSTRUC which displays a listing of the currently defined IGORE structures.

STRUCTYPE

This application returns the IGORE structure type of the passed record pointer. The format of this command is:

Calling Sequence

structure_name = STRUCTYPE(r_pointer)

Arguments

R_POINTER
An IGORE record pointer, or single element of a pointer array, for which the structure type is requested.

Discussion

This routine returns a character string indicating the structure name of the passed IGORE record pointer R_POINTER. A null string is returned if the record pointer is invalid.

STUFF_COV

This procedure will load the IGR$COVAR aux information into an sdr array. The format of the command is:

Calling Sequence

STUFF_COV, sdr, cov

[This procedure is written in IDL]

Arguments

SDR

An input/output SDR record into which the covariance data (IGR$COVAR) records is to be placed.

COV

An input IDL structure which maps IGR$COVAR aux structure.

SUMDET

This procedure reads SDB files output from DAY_SUMMER and sums daily records by detector to create one SDR of spectral data form each detector. If the keyword ALLDETS is set, then all detector data will be summed together into one SDR. The format of this command is:

Calling Sequence

SUMDET, infile, sumsdr

[This procedure is written in IDL]

Arguments

INFILE

A character string specifying the name of the SDB file containing the daily records of the source data being analyzed. This file should be the output file from a previous call to DAY_SUMMER.

SUMSDR

SDR array containg summed spectral data by detector.

Keywords

INSDR

An input SDR array containg daily records of the source data being analyzed and which, if present and nonzero, will be used instead of the input SDB file name specified by the arugment INFILE.

ALLDETS

If present and nonzero, all detector data will be summed into one SDR.

OUTPUTSDB

A character string defining the file name for an output SDB file.

RESOLUTION

An input scalar defining the angular resolution in degrees for identifying unique configurations. When present and nonzero, it will call the routine CRC_ROUNDER to return only unique Compound Response Coefficients (CRC's).

Discussion

Related Routines

Also see the related function CRC_ROUNDER.

TEXT_SPECUNITS

This function decodes the value of the SDB header variable HDR.SPECUNIT, returning a text string. The format of this command is:

Calling Sequence

string = TEXT_SPECUNITS, r_sdr

Arguments

R_SDR
Input SDR (or SDR array) for which spectral units will be displayed. Not required if UNITS is present.

Keywords

UNITS
Optional input integer containing a spectral units bit field, in the format of HDR.SPECUNIT. Ignored if R_SDR also present.

Discussion

This function decodes the value of the SDB header variable HDR.SPECUNIT, returning a text string (or string array). See example below.

```
Igore> print,text_specunits(sdr)
 COUNTS  (DTW corrected)  /  SEC  MEV
```

TIMER

This procedure provides user control of IGORE timing utilities to gather timing statistics on processing. The format of this command is:

Calling Sequence

TIMER

[This procedure is written in IDL]

Arguments

NONE

Keywords

/INIT
If present and non-zero, /INIT specifies that the system timing statistics are to be initialized.

Discussion

This routine initializes and displays various process statistics. The default action of TIMER is to display the process statistics. The system timer must be initialized using the /INIT keyword prior to displaying the process statistics. An example of the usage of TIMER is given below:

```
Igore> TIMER, /init
Igore> .RUN test1
Igore> TIMER
 ELAPSED:    0 00:14:39.38  CPU: 0:00:05.14  BUFIO: 68  DIRIO: 0  FAULTS: 0
Igore> TIMER, /init
Igore> .RUN test2
Igore> TIMER
 ELAPSED:    0 00:04:39.38  CPU: 0:00:01.14  BUFIO: 68  DIRIO: 0  FAULTS: 0
```

TJDTODATE

This function converts the specified Truncated Julian day into an ASCII character string. The format of this command is:

Calling Sequence

date = TJDTODATE(tjd)

[This function is written in IDL]

Arguments

TJD
The Truncated Julian day, including fractional day, for which the ASCII date is to be determined.

Discussion

This function returns in DATE the character string date corresponding to the specified value of TJD. The form of DATE is "dd-mmm-yyyy hh:mm:ss.ss".

TJD_ASCII

This function converts the specified Truncated Julian day into an ASCII character string. The format of this command is:

Calling Sequence

date = TJD_ASCII(tjd)

[This function is written in IDL]

Arguments

TJD
The Truncated Julian day, including fractional day, for which the ASCII date is to be determined.

Keywords

/NOYEAR
If present and non-zero, /NOYEAR specifies that DATE does not include the year or day (*e.g.*, only "hh:mm:ss" is returned in DATE).

/MSEC
If present and non-zero, /MSEC specifies that DATE includes the time to an accuracy of 0.1 milliseconds (*e.g.*, "yy/ddd hh:mm:ss.ssss" is returned in DATE).

Discussion

This function returns in DATE the character string date corresponding to the specified value of TJD. The form of DATE is "yy/ddd hh:mm:ss".

TJD_FROM_ASCII

This function converts the specified date character string into the corresponding Truncated Julian day. The format of this command is:

Calling Sequence

tjd = TJD_FROM_ASCII(date)

[This function is written in IDL]

Arguments

DATE

A character string specifying the date for which the truncated Julian day is to be determined. The form of DATE is "yy/ddd hh:mm:ss.sss". Any unspecified fields of DATE are assumed to be zero.

Discussion

This function returns in TJD the Truncated Julian day corresponding to the specified value of DATE.

TJD_FROM_VAXASC

This function converts the specified date character string into the corresponding Truncated Julian day. The format of this command is:

Calling Sequence

tjd = TJD_FROM_VAXASC(date)

Arguments

DATE

A character string specifying the date for which the truncated Julian day is to be determined. The form of DATE is "dd-mmm-yyyy hh:mm:ss.sss". Any unspecified fields of DATE are assumed to be zero.

Discussion

This function returns in TJD the Truncated Julian day corresponding to the specified value of DATE.

TJD_YMDASCII

This function converts the specified Truncated Julian day into an ASCII character string. The format of this command is:

Calling Sequence

date = TJD_YMDASCII(tjd)

[This function is written in IDL]

Arguments

TJD

The Truncated Julian day, including fractional day, for which the ASCII date is to be determined.

Keywords

/NOYEAR

If present and non-zero, /NOYEAR specifies that DATE does not include the year or day (*e.g.*, only "hh:mm:ss" is returned in DATE).

/MSEC

If present and non-zero, /MSEC specifies that DATE includes the time to an accuracy of 0.1 milliseconds (*e.g.*, "yyyy mmm dd hh:mm:ss.ssss" is returned in DATE).

Discussion

This function returns in DATE the character string date corresponding to the specified value of TJD. The form of DATE is "yyyy mmm dd hh:mm:ss".

TJS_ASCII

This function converts the specified Truncated Julian day (expressed in seconds) into an ASCII character string. The format of this command is:

Calling Sequence

date = TJS_ASCII(tjs)

[This function is written in IDL]

Arguments

TJS

The Truncated Julian day, expressed in seconds, for which the ASCII date is to be determined.

Keywords

/NOYEAR

If present and non-zero, /NOYEAR specifies that DATE does not include the year or day (*e.g.*, only "hh:mm:ss" is returned in DATE).

/MSEC

If present and non-zero, /MSEC specifies that DATE includes the time to an accuracy of 0.1 milliseconds (*e.g.*, "yy/ddd hh:mm:ss.ssss" is returned in DATE).

Discussion

This function returns in DATE the character string date corresponding to the specified value of TJS. The form of DATE is "yy/ddd hh:mm:ss".

TPU

This application provides the ability to edit a file directly from the IGORE environment. The format of this command is:

Calling Sequence

TPU [, file_name]

[This procedure is written in IDL]

Arguments

FILE_NAME
An optional character string specifying the name of the file to edit.

Discussion

This procedure provides access to the TPU editor directly from the IGORE environment; using this command is significantly faster than spawning a process to execute the editor. The name of the file to edit is prompted for if the optional argument FILE_NAME is not provided.

TRACE

This procedure is used to specify the IGORE Condition Handler error traceback mode. The format of this command is:

Calling Sequence

TRACE

[This procedure is written in IDL]

Arguments

NONE

Keywords

/DEBUG
If present and non-zero, /DEBUG initiates traceback and enters the debugger on an error condition. This is the default.

/NODEBUG
If present and non-zero, /NODEBUG initiates traceback only on an error condition.

/OFF
If present and non-zero, /OFF disables all traceback and debugging options.

Discussion

This routine provides the ability to dynamically change the IGORE Condition Handler error traceback mode. Care should be taken using the debugger option since an `EXIT` from the debugger will exit the IGORE session.

If no keywords are passed, the default action is /DEBUG. Only one optional parameter should be specified. If more than one is specified, the priority order is /OFF (highest), /DEBUG, /NODEBUG.

TRI_SURF

This routine is similar to MIN_CURVE_SURF but the surface fitted is a smooth surface, not a minimum curvature surface. This routine has the advantage of being much more efficient for larger numbers of points. This function returns a two-dimensional floating point array containing the interpolated surface, sampled at the grid points. The format of this command is:

Calling Sequence

return = TRI_SURF (Z [, X, Y])

[This procedure is written in IDL by RSI]

Arguments

Z

The IDL array containing the Z coordinates of the data points on the surface. Points need not be regularly gridded. For regularly gridded input data, X and Y are not used, but the grid spacing would be specified via the XGRID and YGRID (or XVALUES and YVALUES) keywords, and Z must be a two dimensional array.

X

The IDL array containing the X coordinates of the data points on the surface. Optional for regularly gridded input data. For irregular grids, X must be present and have the same number of elements as Y and Z.

Y

The IDL array containing the Y coordinates of the data points on the surface. Optional for regularly gridded input data. For irregular grids, Y must be present and have the same number of elements as X and Z.

Keywords

EXTRAPOLATE

Input which, if present and non-zero, extrapolates the surface to points outside the convex hull of input points. Has no effect if input points are regularly gridded.

MISSING

Input which, if present and non-zero, sets points outside the convex hull of input points to this value. Default is 0. Has no effect if input points are regularly gridded.

REGULAR

Input which, if present and non-zero, specifies that the Z input parameter is a two dimensional array of dimensions (N,M), containing measurements over a regular grid. If any of XGRID, YGRID, XVALUES, YVALUES are specified, REGULAR is implied. REGULAR is also implied if there is only one parameter, Z. If REGULAR is set, and no grid (_VALUE or _GRID) specifications are present, the respective grid is set to (0, 1, 2, ...).

XGRID

Input IDL two dimensional array, [XSTART, XSPACING], defining the input grid in the X direction. Do not specify both XGRID and XVALUES.

XVALUES

Input IDL array where XVALUES(i) contains the X location of Z(i,j). XVALUES must be dimensioned with N elements.

YGRID

Input IDL two dimensional array, [YSTART, YSPACING], defining the input grid in the Y direction. Do not specify both YGRID and YVALUES.

YVALUES

Input IDL array where YVALUES(i) contains the Y location of Z(i,j). YVALUES must be dimensioned with N elements.

GS

If present, GS must be a two-element vector [XS, YS], where XS is the horizontal spacing between grid points and YS is the vertical spacing. The default is based on the extents of X and Y. If the grid starts at X value Xmin and ends at Xmax, then the default horizontal spacing is (Xmax - Xmin)/(NX-1). YS is computed in the same way. The default grid size, if neither NX nor NY is specified, is 26 by 26.

BOUNDS

If present, BOUNDS must be a four element array containing the grid limits in X and Y of the output grid: [Xmin, Ymin, Xmax, Ymax]. If not specified, the grid limits are set to the extent of X and Y.

NX

The output grid size in the X direction. NX need not be specified if the size can be inferred from GS and BOUNDS. The default value is 26.

NY

The output grid size in the Y direction. See NX.

Discussion

This function interpolates either a regularly or irregularly gridded set of points with a C1 smooth quintic surface. The built-in IDL routines /TRIANGULATE and /TRIGRID(/QUINTIC) are used. Returned is a two-dimensional floating point array containing the interpolated surface, sampled at the grid points. Examples of the usage of TRI_SURF are given below:

```
;       Regularly gridded cases
```

```
Igore> z = randomu(seed, 5, 6)
Igore> CONTOUR, TRI_SURF(z, /REGULAR)
;       Irregularly gridded cases
;       Make a random set of points that lie on a gaussian:
Igore> n = 15                                ; # random points
Igore> x = RANDOMU(seed, n)
Igore> y = RANDOMU(seed, n)
Igore> z = exp(-2 * ((x-.5)^2 + (y-.5)^2))    ; The gaussian
;       get a 26 by 26 grid over the rectangle bounding x and y:
Igore> r = TRI_SURF(z, x, y)                  ; Get the surface.
;       Or: get a surface over the unit square, with spacing of 0.05:
Igore> r = TRI_SURF(z, x, y, GS=[0.05, 0.05], BOUNDS=[0,0,1,1])
;       Or: get a 10 by 10 surface over the rectangle bounding x and y:
Igore> r = TRI_SURF(z, x, y, NX=10, NY=10)
```

NOTE: The output grid must enclose convex hull of the input points.

VMSTIME_ASCII

This function converts the specified VAX time specification into an ASCII character string. The format of this command is:

Calling Sequence

date = VMSTIME_ASCII(vax_date_spec)

[This function is written in IDL]

Arguments

VAX_DATE_SPEC

A double precision scalar containing the date specification in VMS format.

Keywords

/NOYEAR

If present and non-zero, /NOYEAR specifies that DATE does not include the year or day (*e.g.*, only "hh:mm:ss" is returned in DATE).

Discussion

This function returns in DATE the character string date corresponding to the specified value of VAX_DATE_SPEC. The form of DATE is "yy/ddd hh:mm:ss".

YMD_TO_TJD

This function converts the specified Gregorian date into the corresponding Truncated Julian day. The format of this command is:

Calling Sequence

tjd = YMD_TO_TJD(year, month, day)

[This function is written in IDL]

Arguments

YEAR
An integer value specifying the year (*e.g.*, 1991).

MONTH
An integer value specifying the month (1–12).

DAY
An integer value specifying the day (1 – 31).

Discussion

This function returns in TJD the Truncated Julian day for the specified Gregorian date YEAR, MONTH and DAY.

YYDOY_TO_TJS

This function returns a date in Truncated Julian Seconds for an input date expressed as YYDOY (yearday-of-year) and seconds of day.

Calling Sequence

return = YYDOY_TO_TJS (yydoy [,sod])

Arguments

YYDOY
Input date expressed as year-day-of-year. Day 100 of year 1992 would be 92100.

SOD
Optional input seconds of the day.

Discussion

This function returns a date in Truncated Julian Seconds for an input date expressed as YYDOY (year-day-of-year) and optionally, seconds of day.

Chapter 4

IGORE Quick Reference

Functional Summary

IGORE Record Manipulation (General)

`COPYREC`, *r_source, r_destination*	Copies one IGORE record element to another.
`CREREC`, *structure_type, r_pointers, nrec*	Creates an IGORE record.
`DELREC`*[, r_pointers]*	Deletes specified IGORE record.
`DUPREC`*[, r_source], r_destination*	Duplicates an IGORE record array.
`FLDNAME(`*r_pointer*`)`	Return list of IGORE record field names.
`GETAUX(`*auxkey, field_name, r_sdr [, index]*`)`	Extract field from SDR AUX record.
`GETAUXREC(`*auxkey, r_sdr [, index]*`)`	Extract entire AUX record from SDR.
`GETFLD(`*field_name, r_pointers*`)`	Extract field from IGORE record.
`GETIFLD(`*field_pointer*`)`	Extract data from IGORE indirect field pointer.
`LISTREC`, *r_pointers*	List contents of an IGORE record.
`MERGEREC`, *r_source, r_destination*	Copies IGORE records into single record array.
`MOVEIND`, *in_fld, r_in_ptr, r_out_ptr* `MOVEIND`, *r_in_ptr, out_fld, r_out_ptr* `MOVEIND`, *in_fld, r_in_ptr, out_fld, r_out_ptr*	Copies direct or indirect records from one IGORE record to another.
`NUKEREC`	Deletes all IGORE records.

`POINTERS`, *r_pointers*	Recreate lost pointers for IGORE record.
`PUTAUX`, *auxkey, field_name, r_sdr, value [, index]*	Put value in AUX record of SDR.
`PUTAUXREC`, *auxkey, r_sdr*	Put AUX record into SDR.
`PUTFLD`, *field_name, r_pointer, value*	Put value into IGORE record field.
`REC_EXIST(`*[variable]*`)`	Determine if specified IGORE record exists.
`RECINDX(`*r_pointer*`)`	Return index into parent IGORE record.
`RECNAME(`*r_pointer*`)`	Returns IGORE record name for record pointer.
`RENAME_REC`, *r_old_record, r_new_record*	Rename IGORE record.
`SETUP_REC`, *r_pointer, input_file*	Load IGORE record fields from ASCII file.
`SHOWREC`*[, r_record]*	List current IGORE records.
`STRUCTYPE(`*r_pointer*`)`	Returns IGORE structure type of record pointer.

IGORE Structures

`LOADSTRUC`, *file_name*	Load IGORE structure definition.
`SHOWFLD`*[, structure_name]*	List fields defined in IGORE structure.
`SHOWSTRUC`	List known IGORE structures.
`STRUCEXIST(`*struc_name*`)`	Indicates whether IGORE structure exists.
`STRUCTYPE(`*r_pointer*`)`	Returns IGORE structure type of record pointer.

Saving / Restoring IGORE Records

`RESTORE_REC`*[, save_file]*	Restore saved IGORE records from file.
`SAVE_REC`*[, file[, confirm]]*	Save IGORE records to file.

Observation Description

`CONFUSE`, *lun, att, cat, itargs, scanang, xscanang*	Identifies potential confusing sources.
`FIND_SOURCE(`*targ_name, cat*`)`	Return information about specified source.

`FIND_SOURCE_ID(`*targ_name, cat*`)`	Return target ID for specified source.
`FIND_TARGET_MDP(`*startdate, stopdate, targ_name, state, max_off*`)`	Identify observations of specified source.
`OSSE_COORDS`, *zra, zdc, xra, xdc, tscan, txscan, tra, tdc*	Determine OSSE detector pointing coordinates.
`READ_PLANCAT`, *cat*	Read Target Database.
`SC_ATTITUDE`, *att*	Read Observing Plan database.
`SC_COORDS`, *zra, zdc, xra, xdc, tra, tdc, tscan, txscan*	Determine source position in detector coordinates.
`SHOW_CONFIG`, *attitude, target_cat*	Graphical display of OSSE pointing table.
`SHOW_CMDSTATE`, *states=states* `SHOW_CMDSTATE`, *tjd_start=tjd_times*	Displays OSSE command state information.
`SHOW_MDP_CONFIG`, *mdpfile, target_cat*	Graphical display of OSSE pointing table.
`SHOW_PERIOD`, *attitude, target_cat*	Graphical display of OSSE pointing table.
`SHOW_STATETABLE`, *tables, states=states* `SHOW_STATETABLE`, *tables, tjd_start=tjd_times*	Displays OSSE strategy table information.

Pulsar Processing

`BLD_LC`, *targname, startdate, stopdate*	Automated pulsar light curve generation.
`EFOLD`, *r_efold*	Automated epoch-folding of pulsar data.
`EFOLD$OPEN`, *lun, file_name [, file_status[, read_only]]*	Opens EFOLD file for reading/writing.
`EFOLD$READ`, *lun, r_efold, tag*	Read selected EFOLD records from EFOLD file.
`EFOLD$READALL`, *lun, r_efold*	Read all EFOLD records from EFOLD file.
`EFOLD$WRITE`, *lun, r_efold, tag*	Write EFOLD records to EFOLD file.
`EFOLD_PERIOD(`*r_efold, tjd*`)`	Determine pulsar period at specified TJD.
`LCDISPLAY`, *r_efold [, output_file]*	Display data in EFOLD record.
`LCPOINT`, *r_efold [, output_file]*	Display pointing summary from EFOLD record.
`LCSUM`, *r_efold, lc*	Sum selected light curves from EFOLD record.
`PSF$OPEN`, *lun , file_name [, file_status[, read_only]]*	Open PSF file for read/write.

`PSF$WRITE`, *lun, r_sdr*	Write information to PSF file.

Spectral Access / Manipulation / Processing

`BLD_LOGBINS`, *nbins, lower, upper, edges, widths*	Creates log-spaced bins.
`BLD_QUAD_DIF`, *targname, startdate, stopdate*	Automated spectral analysis processing.
`BLD_SQRTBINS`, *nbins, lower, upper, edges, widths*	Creates square-root-spaced bins.
`DAY_SUMMER`, *infiles, outfile [, status]*	Automated summation of difference spectra.
`ENERGY_CAL(`*ndet, gain_ratio [, fwhm_ratio]*`)`	Determine the spectral channel edge energies.
`MERGECHAN`, *edg1, wid1, edg2, wid2, matrix, oedg2, owid2*	Create matrix for merging spectral channels.
`QUICKLOOK`, *qfiles*	Read and plot spectra from specified files over two energy ranges.
`RESAMPLE`, *edg1, wid1, edg2, wid2, matrix, owid2*	Create matrix for rebinning spectral channels.
`SDB$CLOSE`, *lun*	Close specified SDB file.
`SDB$CLOSEALL`	Close all SDB files.
`SDB$DIR`, *lun*	Perform directory of SDB file.
`SDB$LOOK`, *lun*	Display contents of record from SDB file.
`SDB$HITAG(`*lun*`)`	Return highest tagword in SDB file.
`SDB$LOWTAG(`*lun*`)`	Return lowest tagword in SDB file.
`SDB$NTAG(`*lun*`)`	Return number of tagwords in SDB file.
`SDB$OPEN`, *lun, file_name*	Open SDB file for read/write.
`SDB$READ`, *lun, r_sdr, tag*	Read selected SDB records from SDB file.
`SDB$READALL`, *lun, r_sdr*	Read all SDB records from SDB file.
`SDB$REWRITE`, *lun, r_sdr*	Overwrite existing SDB record in SDB file.
`SDB$WRITE`, *lun, r_sdr, tag*	Write SDR records to SDB file.
`SDR$ADD`, *r_sdr1, r_sdr2, r_out*	Add SDR spectra, element-by element.

`SDR$CREATE`, *r_sdr, ndims*	Create IGORE SDR record.
`SDR$FREE`, *r_sdr*	Free SDR indirect fields.
`SDR$GET`, *r_sdr*	Read SDR fields from SDB file.
`SDR$GET_FIT`, *r_fdr, r_sdr*	Evaluate FDR fit and store in SDR.
`SDR$GETFLD(`*field_name, r_pointers*`)`	Extract field from IGORE SDR record.
`SDR$INIT_HDR`, *r_sdr*	Initialize SDR HDR field.
`SDR$LIST`, *r_sdr*	List fields contained in SDR.
`SDR$NORM`, *r_sdr*	Normalize spectra in SDR.
`SDR$SUB`, *r_sdr1, r_sdr2, r_out*	Subtract SDR spectra, element-by element.
`SDR$SUM`, *r_sdr, r_out*	Sum all spectra in SDR array.
`SDR$UPDHDR`, *r_in, r_out, optype*	Update SDR HDR fields.
`SDR$VIEW`, *r_sdr*	Display contents of SDR fields.
`SDR_ESCALE`, *r_sdr*	Add edges and widths to SDRs.
`SDR_HDR_COMPARE`, *r_sdr1, r_sdr2*	Compare HDR fields from two SDRs.
`SDR_REBIN`, *r_sdr, newedg, newwid*	Rebin spectra stored in SDR.

Spectral Fitting Definition / Storage

`FDB$CLOSE`, *lun*	Close an FDB file.
`FDB$DIR`, *lun*	Perform directory of FDB file.
`FDB$LOOK`, *lun*	Display contents of record from FDB file.
`FDB$OPEN`, *lun [, file_name[, file_status[, read_only]]]*	Opens FDB file for read/write.
`FDB$READ`, *lun, r_fdr [, group[, nrec]]*	Read selected FDB records from FDB file.
`FDB$READALL(`*lun, r_fdr*`)`	Read all FDB records from FDB file.
`FDB$WRITE`, *lun, r_fdr*	Write FDR records to FDB file.
`FDR$CREATE`, *r_fdr, ndims*	Create IGORE FDR record.
`FDR$FROM_NML`, *r_fdr*	Initialize FDR from NAMELIST file.

`FDR$TO_NML`, *r_fdr*	Write FDR definition in NAMELIST file.

Display / Plotting / Labeling

`BINPLOT`, *xedg, xwid, yval, yerr*	Log-Log display of data.
`BINPLOT_LABEL`, *x0, y0, text*	Produce labels on `BINPLOT` display.
`BINPLOT_SYMBOL`, *xent, ycent, sym_size*	Produce plot symbols on `BINPLOT` display.
`SHOW_CONFIG`, *attitude, target_cat*	Graphical display of OSSE pointing table.
`SHOW_MDP_CONFIG`, *mdpfile, target_cat*	Graphical display of OSSE pointing table.
`SHOW_PERIOD`, *attitude, target_cat*	Graphical display of OSSE pointing table.

File Access

`EFOLD$OPEN`, *lun, file_name [, file_status[, read_only]]*	Opens EFOLD file for reading/writing.
`EFOLD$READ`, *lun, r_efold, tag*	Read selected EFOLD records from EFOLD file.
`EFOLD$READALL`, *lun, r_efold*	Read all EFOLD records from EFOLD file.
`EFOLD$WRITE`, *lun, r_efold, tag*	Write EFOLD records to EFOLD file.
`FDB$CLOSE`, *lun*	Close an FDB file.
`FDB$DIR`, *lun*	Perform directory of FDB file.
`FDB$LOOK`, *lun*	Display contents of record from FDB file.
`FDB$OPEN`, *lun [, file_name[, file_status[, read_only]]]*	Opens FDB file for read/write.
`FDB$READ`, *lun, r_fdr [, group[, nrec]]*	Read selected FDB records from FDB file.
`FDB$READALL(`*lun, r_fdr*`)`	Read all FDB records from FDB file.
`FDB$WRITE`, *lun, r_fdr*	Write FDR records to FDB file.
`FILE_INFO`*[, lun]*	List files opened by IGORE.
`IGR$CLOSE`, *lun*	Close a file opened by IGORE.
`IGR$CLOSEALL`	Close all files opened by IGORE.
`PSF$OPEN`, *lun , file_name [, file_status[, read_only]]*	Open PSF file for read/write.

`PSF$WRITE`, *lun, r_sdr*	Write information to PSF file.
`SDB$CLOSE`, *lun*	Close specified SDB file.
`SDB$CLOSEALL`	Close all SDB files.
`SDB$DIR`, *lun*	Perform directory of SDB file.
`SDB$LOOK`, *lun*	Display contents of record from SDB file.
`SDB$HITAG(`*lun*`)`	Return highest tagword in SDB file.
`SDB$LOWTAG(`*lun*`)`	Return lowest tagword in SDB file.
`SDB$NTAG(`*lun*`)`	Return number of tagwords in SDB file.
`SDB$OPEN`, *lun, file_name*	Open SDB file for read/write.
`SDB$READ`, *lun, r_sdr, tag*	Read selected SDB records from SDB file.
`SDB$READALL`, *lun, r_sdr*	Read all SDB records from SDB file.
`SDB$REWRITE`, *lun, r_sdr*	Overwrite existing SDB record in SDB file.
`SDB$WRITE`, *lun, r_sdr, tag*	Write SDR records to SDB file.
`SELECT_FILES(`*file_spec*`)`	Select files from wildcard generated list.

Editing

`EDT`*[, file_name]*	Enter EDT editor from IGORE.
`TPU`*[, file_name]*	Enter TPU editor from IGORE.

Programming

`ERRLOG`	Activate IGORE Condition Handler error logging.
`FREEVM`	Free IGORE temporary VM.
`IGORE_DATE`	Print IGORE image date.
`IGRHELP`*[, topic]*	Access IGORE HELP facility.
`SHOWIGORE`	Provides detailed listing of IGORE session.
`TIMER`	Initialize/display process timing statistics.

`TRACE`	Set IGORE Condition Handler error traceback mode.

Mathematical Functions

`ANGSEP(`*lon1, lat1, lon2, lat2*`)`	Separation angle between two vectors.
`BLD_LOGBINS`, *nbins, lower, upper, edges, widths*	Creates log-spaced bins.
`BLD_SQRTBINS`, *nbins, lower, upper, edges, widths*	Creates square-root-spaced bins.
`CROSS_PROD`, *lon1, lat1, lon2, lat2, lon3, lat3*	Vector cross-product of two vectors.
`DISTINCT(`*array*`)`	Returns distinct elements of array.
`ERF(`*value*`)`	Calculate the error function.
`ERFCC(`*value*`)`	Calculate the complementary error function.
`EULER`, *lonin, latin, lonout, latout [, select]*	Astronomical coordinate transformations.
`GAUSS_INT(`*mean, sigma, lower, upper*`)`	Integrate gaussian over specified region.
`SOLAR_RADEC`, *tjd, sol_ra, sol_dec*	Calculates solar coordinates for specified day.

Date Conversions

`DAYCNV`, *jd, year, month, day, hour*	Converts JD to calendar date.
`DAY_OF_YEAR(`*year, month, day*`)`	Determines day-of-year for calendar date.
`TJDTODATE(`*tjd*`)`	Convert TJD to "dd-mmm-yyyy hh:mm:ss.ss".
`TJD_ASCII(`*tjd*`)`	Convert TJD to "yy/ddd hh:mm:ss.ss".
`TJD_FROM_ASCII(`*date*`)`	Convert "yy/ddd hh:mm:ss.ss" to TJD.
`TJD_FROM_VAXASC(`*date*`)`	Convert "dd-mmm-yyyy hh:mm:ss.ss" to TJD.
`TJD_YMDASCII(`*tjd*`)`	Convert TJD to "yyyy mmm dd hh:mm:ss.ss"
`TJS_ASCII(`*tjs*`)`	Convert TJS to "yy/ddd hh:mm:ss.ss"
`VMSTIME_ASCII(`*vax_date_spec*`)`	Convert VAX date to "yy/ddd hh:mm:ss.ss"
`YMD_TO_TJD(`*year, month, day*`)`	Convert calendar date to TJD.

Chapter 5

IGORE Structures

SDR Structure

```
Structure /IGR$SDR/
    Character*20         FILE
    Integer*4            TAG
    Integer*2            MODIFIED
    Record /IGR$SDB_HDR/ HDR
    Indirect_Field       DAT
    Indirect_Field       ERR
    Indirect_Field       EDG
    Indirect_Field       AUX0
    Indirect_Field       AUX1
    Indirect_Field       AUX2
    Indirect_Field       AUX3
    Indirect_Field       AUX4
    Indirect_Field       AUX5
    Indirect_Field       AUX6
    Indirect_Field       AUX7
    Byte                 FLAGS
    Byte                 DAT_KEY
    Byte                 ERR_KEY
    Byte                 EDG_KEY
    Byte                 AUX_KEY(8)
    Integer*4            SEQ_ID
    Integer*2            DIM(2)
    Real*8               TOP_LEVEL_PTR
End Structure
```

FDR Structure

```
Structure /IGR$FDR/
    Character*20         FDB_FILE
    Integer*4            FDB_TAG
    Byte                 FDB_SPEC_NUMBER
    Byte                 FDB_RANGE_NUMBER
    Integer*2            IDET
```

```
    Character*20        SDB_FILE
    Integer*4           SDB_TAGWORD
    Integer*2           SPECSRC
    Integer*2           SPECUNIT
    Integer*4           TARGID
    Character*20        MODEL_ID
    Byte                NPARAM
    Byte                NCONT
    Byte                NLINES
    Integer*2           SETUP_TAG
    Byte                SPECRANGE
    Integer*2           FITMIN
    Integer*2           FITMAX
    Real*4              ENERGYMIN
    Real*4              ENERGYMAX
    Integer*2           MIT
    Integer*4           PLOTMIN
    Integer*4           PLOTMAX
    Real*4              EPS1
    Real*4              EPS2
    Real*4              FLAM
    Real*4              FNU
    Byte                NW
    Integer*2           NCORR
    Integer*2           NDELPTS
    Real*4              CHI2
    Integer*4           NDOF
    Byte                FITSTOP
    Integer*4           FITTJD
    Record /IGR$SDB_HDR/ SDB_HDR
    Indirect_Field      RCORR
    Indirect_Field      IDELPTS
    Indirect_Field      P
    Indirect_Field      DP
    Indirect_Field      MDP
    Indirect_Field      IP
    Indirect_Field      PLB
    Indirect_Field      PUB
    Indirect_Field      PDIFF
    Indirect_Field      DCHI2
    Indirect_Field      PARAMCOMMENT
    Indirect_Field      IERR
    Indirect_Field      SIEVE
    Indirect_Field      GROUPNUMBER
    Real*8              TOP_LEVEL_PTR
End Structure
```

EFOLD Structure

```
Structure /IGR$EFOLD/
    Integer*4       STRT_PB5
```

```
Integer*4       STOP_PB5
Character*80    SDB_FILE
Integer*4       SDB_TAG
Character*20    PROCDATE
Character*80    PDF_FILE
Character*20    PDF_DATE
Character*80    SELQUERY
Character*80    SEL_FILE
Character*80    SCANFILE
Integer*2       EFLDVERS
Character*20    EFLDDATE
Character*12    PSR_NAME
Real*4          PSR_RA
Real*4          PSR_DEC
Real*8          EPH_TJD0
Real*8          EPH_SOD0
Real*8          EPH_PHI0
Real*8          EPH_F0
Real*8          EPH_F1
Real*8          EPH_F2
Real*8          EPH_DM
Real*8          EPH_FREQ
Real*8          BNRY_A1
Real*8          BNRY_E
Real*8          BNRY_T0
Real*8          BNRY_PER
Real*8          BNRYDPDT
Real*8          BNRY_OMZ
Real*8          BNRYDODT
Real*8          BNRY_GAM
Integer*2       NPHASBIN
Integer*2       DPTBL
Byte            MOTION
Integer*2       DSA(4, 8)
Real*4          DET_RA(4, 8)
Real*4          DET_DEC(4, 8)
Real*4          COLLPOSANG(4, 8)
Byte            ACT_PT(4, 8)
Byte            CUR_PT(4)
Integer*2       ACT_WND
Integer*2       WINDOW_LC(8)
Character*4     INSTMODE
Real*4          TIME_RES
Integer*2       MAX_EBE
Byte            SAMPBITS
Byte            INTBYTES
Integer*2       WINDOW_CHMIN(4, 8)
Integer*2       WINDOW_CHMAX(4, 8)
Integer*2       WINDOW_RANGE(4, 8)
Integer*2       WINDOW_SHIFT(4, 8)
Real*4          GAIN(4)
Integer*4       NPKTS
```

```
    Integer*4       NPKTSOVR
    Integer*4       NIER
    Integer*4       NIEROVR
    Indirect_Field  LIGHT_CURVE
    Indirect_Field  LIVE_TIME
    Real*8          TOP_LEVEL_PTR
End Structure
```

SDR AUX Structures

AUDEF Structure

```
Structure /IGR$AU_DEF/
    Byte            SRC_ENTRY(8, 4)
    Byte            BKG_ENTRY(8, 4)
    Byte            DPTBL
    Integer*2       MAX_PTS_PER_POS(8, 4)
    Integer*2       MIN_PTS_PER_POS(8, 4)
    Integer*2       MINBKG(8, 4)
    Real*4          SRC_DT(4)
    Real*4          BKG_DT(8, 4)
    Byte            SEARCH_FLAG(8, 4)
    Character*20    MODEL_TYPE
End Structure
```

AUID Structure

```
Structure /IGR$AUID/
    Character*20    FILE
    Integer*4       TAG
    Real*4          AIJ
    Byte            FDET
    Byte            AUIDMAP(29)
End Structure
```

CAL Structure

```
Structure /IGR$CAL/
    Integer*2       RANGE
    Real*4          WERROR
    Real*4          CENTROID
    Real*4          WIDTH
    Real*4          CERROR
    Real*4          ENERGY
    Character*6     CALDESC
    Character*20    CALMODEL
    Real*4          CALCHI2
    Integer*2       CALNDOF
End Structure
```

COV Structure

```
Structure /IGR$COVAR/
    Character*20    FILE
    Integer*4       TAG
    Real*4          SUMAIJ2
    Real*4          SUMBIJ2
    Real*4          SUMAIJBIJ
    Real*4          CORR
    Byte            DETID
    Integer*4       STRTTIM
    Integer*4       STOPTIM
    Byte            DPTBL
    Integer*2       DSA
    Real*4          ATTZRA
    Real*4          ATTZDC
    Real*4          ATTXRA
    Real*4          ATTXDC
    Integer*4       TARGID
    Real*4          TARGSCAN
    Real*4          TARGXSCAN
End Structure
```

CRC Structure

```
Structure /IGR$CRC/
    Real*4          AIJ
    Byte            DETID
    Integer*4         TJS
    Real*4          LTL
    Real*4          LTM
    Real*4          LTH
    Byte            DPTBL
    Byte            FDET
    Integer*2       DSA
    Byte            SRCFLG
    Real*4          GRORA
    Real*4          GRODC
    Real*4          ATTZRA
    Real*4          ATTZDC
    Real*4          ATTXRA
    Real*4          ATTXDC
    Integer*4       TARGID
    Real*4          TARGSCAN
    Real*4          TARGXSCAN
    Character*8     CMDSTATE_ID
End Structure
```

DIR Structure

```
Structure /IGR$DIR/
```

```
    integer*4       BATSEID
    character*10    DATATYPE
    byte            AUXTYPE
    integer*4       STRTTJS
    integer*4       STOPTJS
    integer*4       STRTTAG
    integer*4       STOPTAG
    character*10    XRAYCLASS
End Structure
```

GEN_HDR Structure

```
Structure /IGR$GEN_HDR/
    Character*8     FILE_KEY       ! 0
    Integer*2       REC_SEQ        ! 0
    Byte            ARRAY_TYPE     ! 2
    Byte            SPAREKEY       ! 3
    Integer*4       TAGWORD        ! 4
    Integer*2       SIZEHDR        ! 8
    Integer*2       NBYTES_PER_RECORD      ! 10
    Integer*2       NUMREC         ! 12
    Integer*2       NUMENT         ! 14
    Integer*2       SIZEENT        ! 16
    Integer*2       DATAVERSION    ! 18
End Structure
```

MULTI Structure

```
Structure /IGR$MULTI/
    Integer*4       STRTTIM
    Integer*2       STRT_FRAC
    Real*4          LTL
    real*4          LTM
    Real*4          LTH
    Integer*2       MLL(4)
    Integer*2       MLM(4)
    Integer*2       MLH(4)
    Integer*2       S1LD(8)
    Integer*2       S2LD(8)
    Integer*2       S3LD(8)
    Integer*2       S4LD(8)
    Integer*2       DTL
    Integer*2       DTM
    Integer*2       DTH
    Integer*2       DAT(544)
End Structure
```

MULTI_HDR Structure

```
Structure /IGR$MULTI_HDR/
    Integer*2       NCHAN
```

```
    Integer*2       NCHANL
    Integer*2       NCHANM
    Integer*2       NCHANH
    Real*4          ACCUMTIM
    Real*4          TSAC
    Integer*2       SPECFLGS
End Structure
```

NOTE Structure

```
Structure /IGR$NOTE/
    Character*80    NOTE
End Structure
```

ORB Structure

```
Structure /IGR$ORB/
    Integer*4       ORBIT
    Integer*4       ORBITSTART
    Integer*4       ORBITSTOP
    Real*4          ORBITINTEG
    Integer*4       SAAENTRY
    Integer*4       SAAEXIT
    Real*4          SAAINTEG
    Real*4          CPML
    Real*4          CPMH
    Real*4          CPML_SAA
    Real*4          CPMH_SAA
End Structure
```

RELSPEC Structure

```
Structure /IGR$RELSPEC/
    Character*40    FILE
    Integer*4       TAGWORD
    Character*40    COMMENT
End Structure
```

RMDEF Structure

```
Structure /IGR$RMDEF/
    Integer*4       KEY
    Integer*4       VERSION
    Integer*4       NCHAN
    Integer*4       NPHOT
    Integer*4       NDET
    Integer*4       DETID(4)
    Real*4          SHDISC(4)
    Real*4          SHFWHM(4)
    Real*4          PSDFRAC(4)
    Real*4          CSIGAIN(4)
```

```
    Real*4          RGAIN(4)
    Real*4          RFWHM(4)
End Structure
```

Auxiliary Data Types

Keyword	Bit	No.	S/D	Class	Description
AUX1					
DAT	0	0	D	n/a	Spectrum data
ERR	1	1	D	1	Spectrum statistical uncertainties (sym. or positive)
NERR	2	2	D	1	Spectrum negative statistical uncertainties
SYERR	3	3	D	1	Spectrum systematic uncertainties (sym. or positive)
NSYERR	4	4	D	1	Spectrum negative systematic uncertainties
EDG	5	5	S	1	Channel lower edge energies
DEDG	6	6	S	1	Channel lower edge energy uncertainties
CEN	7	7	S	1	Channel representative energies
DCEN	8	8	S	1	Channel representative energy uncertainties
WID	9	9	S	1	Channel widths
DWID	10	10	S	1	Channel width uncertainties
PSDEFF	11	11	S	1	PSD efficiency
PSDDEFF	12	12	S	1	PSD efficiency uncertainties
RES	13	13	S	1	Detector resolution
RERR	14	14	S	1	Detector resolution uncertainties
MSDSF	19	19	D	3	Multi-spectrum data from DSF (IGR$MULTI_HDR,IGR$MULTI)
MSDAT	20	20	S	1	Multi-spectrum data
MSERR	21	21	S	1	Multi-spectrum statistical uncertainties (sym. or positive)
NMSERR	22	22	S	1	Multi-spectrum negative statistical uncertainties
MSSYERR	23	23	S	1	Multi-spectrum systematic uncertainties (sym. or positive)
NMSYERR	24	24	S	1	Multi-spectrum negative systematic uncertainties
CALIB	25	25	S	2	Calibration line information (IGR$CAL)
RINFO	26	26	S	1	Detector resolution information
PSDMOD	27	27	S	1	PSD model information
RELSPEC	28	28	S	2	Related spectra (IGR$RELSPEC)
ORBHIST	29	29	S	2	Orbit history (IGR$ORB)
NOTE	30	30	D	2	Footnote (textual comment) (IGR$NOTE)
AUX2					
BKG	0	32	D	1	Estimated background
DBKG	1	33	D	1	Estimated background uncertainties
AUDEF	2	34	S	2	Analyzable unit definition (IGR$AU_DEF)
AUID	3	35	D	2	Analyzable unit identification (IGR$AUID)
SPECLST	4	36	S	1	Selected spectra list
CRC	5	37	D	2	Compound response coefficients (IGR$CRC)
RMTRX	6	38	S	1	Instrument response matrix
PRV	7	39	S	1	Photopeak response vector
MCDB1	8	40	D	1	Reserved for Monte Carlo DB
MCDB2	9	41	D	1	Reserved for Monte Carlo DB
DIR	10	42	S	2	Directory (IGR$DIR)
RMDEF	11	43	S	2	Instrument response matrix definition. (IGR$RMDEF)
RMVEC	12	44	S	1	Instrument response matrix vector.
COV	13	45	D	2	Covariance Information. (IGR$COVAR)

Table 5.1: Auxiliary Data Types: S refers to "Sparse" data types while D refers to "Dense" data types. Class 1 is simple array, class 2 is structured data, class 3 is structured header plus array or structured data. The description includes the IGORE structure name, where applicable.

SDB Header Operations

SDB V7 Header		Process				
SDB Variable	Description	PDA	Best	Write	Sum	Sub
nbytes_per_record	size of data records	r	r	r	r	r
numrec	Number of records	r	r	r	r	r
nument	Number of entities	r	r	r	r	r
sizeent	Number of bytes per entity	r	r	r	r	r
dataversion	Data type version number	r	r	r	r	r
hdrvers	Header version number	u	u		1	1
hdrrev	Header revision number	u	u		1	1
softindex	Creation software index	u	u		1	1
softmajor	Creation s/w major version	u	u		1	1
softminor	Creation s/w minor version	u	u		1	1
cmdstate_id	Cmd state table ID	b	S		b	1
detcfg_id	Det cfg table ID	b	S		b	1
mdpostbl_id	MD Pos. table ID	b	S		b	1
credate	Creation date/time, VAX format	u	u		u	u
detid	Detector ID	u	S		or	1
targid	Target name code	u	S		1	1
targoff	Source/background offset	u	S		1	1
tlmid	Telemetry format ID	u	S		1	1
reactor	Reactor event code	u	or		or	or
dsfstat	DSF stop/event status code	u	or		or	or
dqhaz	Data quality flags hazardous	u	or		or	or
dqinfo	Data quality flags info	u	or		or	1
usrscrnflg	User spectrum screening flags		or		or	or
specsrc	Spectrum source code	u	S		1	1
speccat	Spectral category code	u	S		1	1,u
specunit	Units code	r	r		1,r	1,r
specflgs	Spectrum flags	r	r	r	1,r	1,r
calrpha	CALPHA/RPHA Mux	u			1	1
ltl	SPM low live time	u	S		$\sum$	1
ltm	SPM med live time	u	S		$\sum$	1
lth	SPM high live time	u	S		$\sum$	1
accumtim	Accumulation time	u	S		$\sum$	1
strttim	Start time (in TJS)	u	S		min	1
strt_frac	Start time fraction	u	S		min	1
stoptim	Stop time (in TJS)	u	S		max	1
stop_frac	Stop time fraction	u	S		max	1
orbit	Orbit number	u	S		min	1

SDB V7 Header (continued)		Process				
SDB Variable	Description	PDA	Best	Write	Sum	Sub
srcct(2)	TLM Source Cnt	u	S		1	1
nchan	Total # of spectrum channels	r	r	r	1,r	1,r
nchanl	# of low range channels	r	r	r	1,r	1,r
nchanm	# of medium range channels	r	r	r	1,r	1,r
nchanh	# of high range channels	r	r	r	1,r	1,r
nchan1	# of channels, parameter 1	r	r	r	1,r	1,r
nchan2	# of channels, parameter 2	r	r	r	1,r	1,r
auxflg1	AUX field flags (1st 32)	r	r	r	0,r	0,r
auxflg2	AUX field flags (2nd 32)	r	r	r	0,r	0,r
dptbl	Active positioning table	u	S		1	1
fdet	Detector positioning status	u	S		1	1
dsa	Detector position, end	u	S		1	1
detra	Detector RA	u	S		1	1
detdc	Detector declination	u	S		1	1
targscan	Target latitude (scan angle)	u	S		1	1
targxscan	Target longitude (transscan)	u	S		1	1
earthaz(2)	Earth azmiuth in det. coord.	ss	S		mm	1
earthsep(2)	Earth/collimator seperation	ss	S		mm	1
targra	Target RA 2000 epoch	u	S		1	1
targdc	Target declination 2000 epoch	u	S		1	1
collposang	Collimator position angle	u	S		1	1
sig_scan	Mean scan ptg error (sigma)	0				
sig_xscan	Mean xscan ptg error (sigma)	0				
rgain	Actual Relative Gain Ratio	u	S		1	1
rfwhm	Actual Relative Full Width Half Max	u	S		1	1
stdrgain	Standard Relative Gain Ratio	u	S		1	1
stdrfwhm	Standard Relative Full Width Half Max	u	S		1	1
sum_tim	Rate collection time	u	S		$\sum$	1
"det rate sums"	26 det rate sums (counts)	u	S		$\sum$	1

SDB V7 Header (continued)		Process				
SDB Variable	Description	PDA	Best	Write	Sum	Sub
mcl(2)	McIlwain L	ss	S		mm	1
mcb(2)	McIlwain B	ss	S		mm	1
rigid(2)	Rigidity	ss	S		mm	1
brad(2)	Radial component of mag fld vec	ss	S		mm	
bpol(2)	Polar component of mag fld vec	ss	S		mm	
baz(2)	Az component of mag fld vec	ss	S		mm	
day_night	Day/Night code	u	S		or	1
cpml_rate	CPM low rate	u	S		A	1
cpmh_rate	CPM high rate	u	S		A	1
orbit_tim	Accum time from strt of orb (sec)	u	S		1	1
cpml_orbit_sum	CPML dose since strt of orb (cnt)	u	S		1	1
cpmh_orbit_sum	CPMH dose since strt of orb (cnt)	u	S		1	1
last_saa_exit	Time since last SAA exit (sec)	u	S		1	1
last_saa_tim	CPM accum time for last SAA (sec)	u	S		1	1
cpml_last_saa	CPM low dose during last SAA (cnts)	u	S		1	1
cpmh_last_saa	CPM high dose during last SAA (cnts)	u	S		1	1
grora(2)	RA of s/c	ss	S		mm	1
grodc(2)	Dec of s/c	ss	S		mm	1
grolg(2)	Longitude of sub-s/c point	ss	S		mm	1
attzra(2)	Axis RA 2000 epoch	ss	S		mm	
attzdc(2)	Z Axis Dec 2000 epoch	ss	S		mm	
attxra(2)	X Axis RA 2000 epoch	ss	S		mm	
attxdc(2)	X Axis Dec 2000 epoch	ss	S		mm	
grorad(2)	Alt of s/c above earth	ss	S		mm	
rsc(3)	Barycenter vectors (sec)	u	S		1	1
etut	Barycenter time	u	S		1	1
strtbilvl(5)	Bilevels 0-4, start of acq.	u	S		1	1
endbilvl(5)	Bilevels 0-4, end of acq.	u	S		1	1
ptmp1	PMT mount plate temperature	u	S		1	1
c6tmp	Coll./Co-60 temperature	u	S		1	1
s1tmp	Xtal housing cover temperature	u	S		1	1
cpdtm	CPD temperature	u	S		1	1
"det analogs"	Various analog housekeeping (30)	u	S		1	1
caltw	Calibration tagword	u	C	r	1	1
stdcaltw	Standard calibration tagword	u	C	r	1	1
orbittw	Orbit history tagword	u	C	r	1	1
relspectw	Associated spectra tagword			1	1	1
notetw	Footnote tagword			1	1	1
resptw	Response tagword			1	1	1

SDB V7 Header (continued)		Process				
SDB Variable	Description	PDA	Best	Write	Sum	Sub
achd(25)	Acquisition header	u			1	1
hskptim	Time of slot header packet	u			1	1
tsac	Seconds/TSAC	u	S		1	1
am	Average pot main	u	S		1	1
ar	Average pot redn	u	S		1	1
effecarea	Effective area	u	S		1	1
divisor	Divisor					
phbinnum	Phase bin number					
numphbin	Total number of phase bins					
reserve(37)	Reserve					

Notes:

$\sum$: value is sum of associated headers
0: values initialized to zero
1: values copied from 1st header of array
b: value at the start time of the collection interval
C: values from "source" header, data copied from input to output
min: values copied from header with minimum time
max: values copied from header with maximum time
mm: 1st is min as defined above, 2nd is max as defined above
or: value is logical "or" of associated headers
r: required for correct structure intrepretation or analysis (implies "u")
S: values copied from "source" header
ss: elements are start and stop of interval
u: updated or assigned by process

Chapter 6

Unsupported Applications

This section describes IGORE applications which are currently available, but are no longer being supported. They are still available from IGORE, but may be removed without notice. The use of these routines is not suggested.

FDR$ADJUST_BKG

This procedure provides the ability to adjust the background model type and/or order defined in an FDR. The format of this command is:

Calling Sequence

FDR$ADJUST_BKG, r_fdr [, new_model]

Arguments

R_FDR
An IGORE FDR pointer array or subarray for which the background model type and/or order is to be changed.

NEW_MODEL
An optional character string specifying the new model type and order to be inserted into each element of R_FDR.

Discussion

The form of the optional parameter NEW_MODEL is MMMM## where MMMM is the model type (currently supported types are 'POLY', 'PLAW', 'EXPO', and 'CHEB') and ## is the order of the background model (*e.g.* NEW_MODEL = 'POLY05' indicates a polynomial background model of 4^{th} order having 5 coefficients). If a valid NEW_MODEL is specified, all elements of the passed FDR array are modified to have the same background model type and order. If an invalid NEW_MODEL is specified, or if the parameter is absent, the application will display the current model type and order for each element of the passed FDR array and prompt the user for a new background model specification.

FDR$ADJUST_GAIN

This procedure provides the ability to adjust the peak positions and fit ranges defined in an IGORE FDR for slight changes in detector gain. The format of this command is:

Calling Sequence

FDR$ADJUST_GAIN, r_fdr, gain_shift

Arguments

R_FDR
An IGORE FDR pointer array or subarray for which the peak positions and fit ranges are to be adjusted.

GAIN_SHIFT
A real variable specifying the relative magnitude of the gain shift.

Discussion

The new peak and fit range channels are calculated from the old peak and fit range channels and the parameter GAIN_SHIFT by the following relation:

$$NewChan = (OldChan - IOFF) \times GAIN\_SHIFT + IOFF$$

where *IOFF* is 6 or 261, depending on whether the peak is in the Low or Medium range, respectively.

NOTE: **This routine only supports fit definitions in units of channels (*e.g.* it does not support spectral fits defined in energy units) and has not yet been completely tested!**

INTGRAPH

This application provides an interface to the interactive fitting routine INTERACTIVE_GRAPHICS for IGORE variables. The format of this command is:

Calling Sequence

INTGRAPH, x, y, dy, dx

Arguments

X
Corresponds to the independent variable.

Y
Corresponds to the dependent variable.

DY
The uncertainty of Y.

DX
The uncertainty of X.

Discussion

The array sizes of all passed parameters must be the same. All four parameters are required. No information is directly returned to IGORE.

INTGRAPH_SETUP

This application provides an interface to the interactive graphics setup routine INTGRAPH_SETUP. The format of this command is:

Calling Sequence

INTGRAPH_SETUP

Discussion

There are no parameters for this application. This application is useful if it is desired to re-direct INTERACTIVE_GRAPHICS plotting to another device.

SDB$CLOSEALL

This procedure closes **all** SDB files currently open. The format of this command is:

Calling Sequence

SDB$CLOSEALL

Discussion

This routine will close all open SDB files and free the corresponding logical unit numbers from the logical unit table, where necessary.

The preferred method for closing SDB files is using the command IGR$CLOSE.

SDB6$OPEN

This procedure provides a method for opening an SDB version 6 file for access by SDB6$READ. **Note that SDB version 6 file access is not completely supported by all IGORE applications and their use is not suggested!** The format of this command is:

Calling Sequence

SDB6$OPEN, lun [, file_name [, status [, read_only]]]

Arguments

LUN
An integer specifying the logical unit number to which the SDB file is to be attached.

Optional arguments:

FILE_NAME
A character string specifying the name of the SDB file to be opened (up to 80 characters).

STATUS
A character string specifying the status with which the SDB file is to be opened. Supported values are 'NEW', 'OLD', and 'UNKNOWN'. The default value is 'OLD'.

READ_ONLY
A logical flag which, if set, specifies that the SDB file is to be opened with READ-ONLY protection. This is the default mode for files opened with a STATUS of 'OLD' or 'UNKNOWN'.

Discussion

If the logical unit specified by LUN is already attached to a file, an error shall be issued and the application aborted. If the optional parameter FILE_NAME is not provided, the routine will allow the user to select a file from a directory listing of a user specified directory.

SDB6$READ

This procedure provides a method of reading data from an SDB version 6 file into an IGORE SDR. **Note that SDB version 6 file access is not completely supported by all IGORE applications and their use is not suggested!** The format of this command is:

Calling Sequence

SDB6$READ, lun, r_sdr, tag

Arguments

LUN
An integer specifying the logical unit number from which the data are to be read.

R_SDR
An SDR pointer array or subarray into which the data are to be loaded. Note that conversion from the SDB version 6 format into the IGORE version 7 SDR is automatically performed.

TAG
A longword scalar or array specifying the tagwords of the SDB records to be read. For non-positive values of TAG, a sequential read is performed and the tagword of the record read is returned in TAG. Note that if a subarray is passed as TAG, sequential reads will be performed but the non-positive TAG elements will **not** be modified to represent the actual tagwords read since IDL passes subarrays as temporary arrays.

Keywords

DAT
Input switch which, if present and non-zero, indicates the DAT AUX field (AUX type 0) is to be loaded into the SDR.

ERR
Input switch which, if present and non-zero, indicates that the ERR AUX field (AUX type 1) is to be loaded into the SDR.

FOOT
Input switch which, if present and non-zero, indicates that the 160 character field in the sdb6 header is to be loaded into the SDR as a footnote AUX field (AUX type 30).

Discussion

The SDB file must be opened using SDB6$OPEN prior to using this application.

If the number of array elements in the arrays R_SDB and TAG are different, a warning shall be issued and only the first N elements shall be read, where N is the minimum number of elements in the R_SDB and TAG arrays.

If a tagword specified by TAG does not exist, a warning shall be issued and the execution shall continue. The element of TAG producing the warning shall be set to 0 and the data associated with the corresponding element of R_SDB shall be deleted.

SDR$CALC_ERR

This procedure estimates the statistical uncertainty in the passed spectra using Poisson statistics on the observed number of counts in each channel. The format of this command is:

Calling Sequence

SDR$CALC_ERR, r_sdr

Arguments

R_SDR
An SDR pointer array or subarray for which the statistical uncertainties are to be determined.

Keywords

MODE
An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 0, indicating that if any indirect fields in the input SDRs are not available, they will be read from the corresponding SDB file and stored in a temporary area.

Discussion

The resultant error array will overwrite the original error array stored in R_SDR, if present. If the units of the spectrum are not COUNTS, a warning message is issued and processing continued with the next element of the pointer array.

The statistical uncertainties represent the 1σ error and are estimated from the number of observed counts in each spectrum channel using Poisson statistics by averaging the range between the upper and lower Poisson error. That is:

$$Error = \frac{1}{2}(UpperError - LowerError)$$

where *UpperError* and *LowerError* represent the Poisson upper and lower bounds of the 1σ range for the observed number of counts. For an observed number of counts ≥ 30, the upper and lower errors are determined by adding or subtracting the square-root of the observed number of counts, respectively.

SDR$DTW_SLICE

This procedure produces a set of 16 time-decay spectra, one for each DTW channel, from an input 2-parameter spectrum. The format of this command is:

Calling Sequence

SDR$DTW_SLICE, r_sdr, r_out

Arguments

R_SDR
An input SDR pointer, or single element of a pointer array, containing 2-parameter data.

R_OUT
An output-only SDR into which the resulting time-decay spectra are to be placed.

Discussion

The resulting time-decay spectra are normalized to 256 channels. A single SDB header is associated with all 16 of the output spectra.

If R_OUT does not exist prior to the call to SDR$DTW_SLICE, it will be created. If the record R_OUT does exist, it will be deleted and recreated with the proper dimension size.

The input record R_SDR is not affected.

SDR$INTGRAPH

This procedure provides an interface for an SDR into the interactive fitting routine INTERACTIVE_GRAPHICS. The format of this command is:

Calling Sequence

SDR$INTGRAPH, r_sdr

Arguments

R_SDR
An input SDR pointer (or single element of a pointer array) containing data to displayed and fitted.

Keywords

SPECUNIT
An integer specifying the spectrum units into which the data from the SDR are to be converted before displaying. The spectrum units are specified in the same way as the SDB header variable SPECUNIT. The default value of SPECUNIT is 0, indicating the spectrum is to be displayed in the same units as it is stored in the SDR.

Discussion

This routine passes the information stored in the R_SDR, including the DAT, ERR and EDG fields (if available), directly to INTERACTIVE_GRAPHICS for interactive manipulation and fitting. If the WID field is available, it is also provided to INTERACTIVE_GRAPHICS; otherwise, the channels are assumed to be continuous and the widths determined from the channel edges.

If an INTERACTIVE_GRAPHICS routine has not been previously executed during the current IGORE session, SDR$INTGRAPH will prompt for the user's terminal type with the following:

```
Based on terminal type, VT100 these default values have been set:
   Graphics terminal, option =   4014     1
Enter new values (<cr> if ok), TERM,OPT:
```

The most common responses to this prompt are:

10,1 – X-Windows terminal.

4014,1 – Terminal supporting standard VT100-style plotting.

If more than a single record element is passed, the first element is displayed. No information is directly returned to IGORE. Note that INTERACTIVE_GRAPHICS can produce a NAMELIST file which may then be read into an FDR using the routine FDR$FROM_NML. This FDR can then be used to fit spectra in IGORE using the routine SDR$SPEC_FIT.

A description of the key commands for INTERACTIVE_GRAPHICS is given in Chapter 7.

The input record R_SDR is not affected.

SDR$SCALE

This procedure provides a method of multiplying a spectrum, channel-by-channel, by the passed scalar value. The format of this command is:

Calling Sequence

SDR$SCALE, r_sdr, xscale

Arguments

R_SDR
An SDR pointer array or subarray for which the data and errors are to be scaled.

XSCALE
The scale factor by which the data and errors from R_SDR are to be multiplied.

Keywords

MODE
An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 1, indicating that if any indirect fields in the input SDR are not available, they will be read from the corresponding SDB file and stored in the SDR.

Discussion

The uncertainties are obtained from the ERR field of R_SDR, if available, or estimated from the number of counts in the original spectrum channel based on Poisson statistics, if possible. The spectrum and errors are and multiplied by the scalar specified XSCALE and stored in the DAT and ERR fields in R_SDR.

SDR$SPEC_FIT

This procedure provides a method for performing spectral fits on groups of spectra. The format of this command is:

Calling Sequence

SDR$SPEC_FIT, r_sdr, r_fdr

Arguments

R_SDR
An SDR pointer array or subarray containing data to be fitted.

R_FDR
An FDR array or subarray containing the definitions of the fits to be performed.

Keywords

MODE
An integer specifying the method of storing the indirect SDR fields, if necessary (see Table 2.1 in Chapter 2). The default value of MODE is 0, indicating that if any indirect fields in the input SDRs are not available, they will be read from the corresponding SDB file and stored in a temporary area.

DUPERFIT
An integer which, if provided, specifies that the fit is to be performed using SUPERDUPERFIT. The value of DUPERFIT indicates the fitting parameter for which SUPERDUPERFIT is to be applied. If DUPERFIT $<$ 0, SUPERDUPERFIT is performed on **all** fit parameters.

CHI2_INC
A real value specifying the χ^2 increment to be used by SUPERDUPERFIT. This keyword is valid only when DUPERFIT is selected. The default value of CHI2_INC is 1.

SPECUNIT
An integer specifying the spectrum units into which the data from the SDR are to be converted prior to performing the spectral fit. The spectrum units are specified in the same way as the SDB header variable SPECUNIT. The default value of SPECUNIT is 0, indicating the input spectral units are not to be modified.

FROMP
Input switch which, if present and non-zero, specifies that the initial guess parameters for the fit are to be obtained from the P array of the previous fit instead of

the INIT_P array. This keyword is valid only when fitting arrays of spectra and the fit definition for one or more of the array elements is not initialized. The default action is for the initial fit parameters to be obtained from the INIT_P array of the previous fit. See the description below for more details.

FITPLOT
Input switch which, if present and non-zero, specifies that a plot of the spectral fit is to be produced for each of the fits.

ERRPLOT
Input switch which, if present and non-zero, specifies that a plot of the fit residuals is to be produced for each of the fits.

PLOT
Input switch which, if present and non-zero, specifies that plots of both the spectral fit and the fit residuals are to be produced for each of the fits.

NOPROMPT
Input switch which, if present and non-zero, specifies that the interactive user prompts are to be disabled. The default is for the interactive user prompts to be displayed.

Discussion

The spectral fits are performed element-by-element on the input arrays R_SDR and R_FDR. The spectrum contained in the i^{th} element of R_SDR is fitted using the fit definition from the i^{th} element of R_FDR. The results of the spectral fit are placed into the i^{th} element of R_FDR. The input record R_SDR is not affected.

The initial values for the fit parameters are taken from the P array of R_FDR. If an element of R_FDR does not contain a valid fit definition, the fit definition from the **previous** element of R_FDR is used. In this case, the initial values for the fit parameters are taken from the INIT_P array of the **previous** R_FDR element, unless the FROMP keyword is specified.

By default, the spectral fits are performed using SUPERFIT and the uncertainties in the fitted parameters are placed in the DP field of R_FDR. If the keyword DUPERFIT is specified, the fitting is performed using SUPERDUPERFIT. In this case, the parameter for which the χ^2 mapping is performed is specified by the value of DUPERFIT and the χ^2 increment used to determine the uncertainties in the fitted parameter is given by the keyword CHI2_INC. The resulting upper and lower uncertainties in the fitted parameters are placed in the DP and MDP fields of R_FDR, respectively.

Plots of the spectral fits and residuals can be generated by setting the appropriate keywords. The plots can be redirected to an output file by specifying the terminal type and option as "4014,3". Use the routine INTGRAPH_SETUP to reinitialize the terminal type and option.

Specifying the keyword NOPROMPT is useful when putting SDR$SPEC_FIT in batch jobs or IDL procedures.

Related Routines

See the related routines SDR$INTGRAPH and FDR$FROM_NML. Also see the routine FFOLD which performs spectral fits using forward-folding techniques.

Note that the use of SDR$SPEC_FIT is being phased out and will be replaced by an IDL fitting routine.

SDR$SPEC_PLOT

This procedure provides a method for producing plots of groups of spectra. The format of this command is:

Calling Sequence

SDR$SPEC_PLOT, r_sdr

Arguments

R_SDR
An SDR pointer array or subarray containing the spectra to be plotted.

Discussion

This routine will produce plots of all of the spectra in R_SDR. The plots can be redirected to an output file by specifying the terminal type and option as "4014,3". Use the routine INTGRAPH_SETUP to reinitialize the terminal type and option.

Note that the use of SDR$SPEC_PLOT will be phased out. Similar plots of spectra stored in an SDR can be easily produced using the IDL `PLOT` or `PLOTERR` commands.

Chapter 7

Interactive Graphics Description

INTERACTIVE_GRAPHICS Cursor Commands

The following command functions are supported by the cursor command input:

1 – Specifiy the MINIMUM values (x,y) of a temporarily defined graphics region to be utilized as part of a subsequent cursor command. Depending on the subsequent command, the x-value, the y-value, or both may be significant.

2 – Specifiy the MAXIMUM values (x,y) of a temporarily defined graphics region to be utilized as part of a subsequent cursor command. Depending on the subsequent command, the x-value, the y-value, or both may be significant.

A – Set scaling to AUTOSCALE for the next refresh of the graphics screen.

B – Set the BACKGROUND level for the initial guess for fitting background. If prior 1,2 commands have been set, the initial background level and 1st polynomial coefficient are estimated, otherwise the background is assumed constant.

C – CONSTRAIN the position of the peak nearest to the cursor x-value to remain within the x-value range specified by prior 1,2 cmds.

D – DELETE the peak nearest to the cursor x-position from the fit model.

E – Plot the ERRORS or differences in the data and the fit model from the previous fit.

F – Perform a least-squares FIT to the data within the specified (L,U) region using the background and peak estimates previously specified. The resulting fit model is displayed and peak positions summarized.

H – No longer supported. See \R.

K – Delete all points in range 1 to 2 inclusive from fit. Deleted points are marked with a circle. If points have x-axis error bars, boundaries of region are determined by the points whose x extent includes the cursor when the 1,2 commands are issued. If there are no x-axis error bars, the nearest points are picked.

L – Set the LOWER bounds of the plotting region to the x-position of the current cursor. This value is used to specify the region displayed when the graphics screen is refreshed. When the plotting region is changed, if part of the fitting region is excluded by the change, the fitting region is redefined to coincide with the plotting region as necessary.

M – Change continuum model for fitting. Waits for second character entry. Second character can be:

- C – Chebyshev polynomial
- E – Exponential continuum p1*exp(p2*x)+p3
- L – Power Law continuum p1*(x/p3)**p2
- P – Polynomial continuum p1+p2*x+p3*x**2....

In each case, a 2-point estimate will be made on the B command if 1 and 2 have been set.

N – Exit the interactive graphics routines to get the NEXT array of data.

O – Return the graphic display to the ORIGINAL data set passed to the interactive graphics call.

P – Define a PEAK at the current cursor position. X-position is initial guess for position. Y-position is used in initial guess of intensity. The initial guess for FWHM is obtained by using the leading edge of the peak. In a region of unresolved peaks, the W command can be used to specify a better initial FWHM estimate.

Q – Enable the QUADRATIC element in the background model. The default model is 1st order polynomial.

R – REFRESH or REDRAW the graphic display utilizing all of the previously specified modifications.

S – SETUP: causes format-parameter namelists to be called. Allows listing of current format parameters, presetting of parameters via /INFILE=, recording of current parameters via /OUTFILE=, or modification of individual parameters.

U – Set the UPPER bounds of the plotting region to the x-position of the current cursor. This value is used to specify the region displayed when the graphics screen is refreshed. When the plotting region is changed, if part of the fitting region is excluded by the change, the fitting region is redefined to coincide with the plotting region as necessary.

V – Remove the constraint in position on the peak nearest to the current cursor x-position. Removes action of C command.

W – Set the WIDTH (FWHM) of the peak nearest to the current cursor x-position to the difference in x-values specified by previous 1,2 commands.

X – EXIT the interactive graphics routines to the calling program.

Y – Set the Y-scale to the y-values specified by previous 1,2 commands. This new scaling will be used in next and all subsequent screen refreshes, until canceled by "A" command.

Z – Zero all fit parameters, zero number of peaks, and return fit region to plot region boundaries.

? – Access the help facility on the cursor commands in the interactive graphics routines.

\A – Move axis label to new position or assign a label from the OTHERLABELS list. Command takes a second character input. The second character can be:

- X – Move X-axis label to cursor position (cursor at lower left corner of label).
- Y – Move Y-axis label to cursor position (label rotated 90 degrees; cursor at lower left corner of label)
- 0-9 – Place label #0-9 at cursor position (cursor at center left edge of label). Labels are input using the S command and assigning character strings of up to 80 characters to the variable OTHERLABELS(0-9).

\C – Assigns color indices to regions. It ignores cursor position and waits for a second and third character input. The second character is a letter indicating the region to define. The third character is a digit from 0-7 specifying the color index for that region. The regions are specified below. The color indices refer to the default colors for the terminal in question.

Cmd Letter	Region Number	Region
D	1	data
G	2	grid and axis labels
F	3	fit curves
T	4	fit output text
L	5	main label
O	6	other labels
A	7	arrows
P	8	deleted point markers
B	9	background

\D – Delete a label. A second input character is required.

- X – Delete x-axis label.
- Y – Delete y-axis label.
- 0-9 – Delete OTHERLABELS #0-9.

\E – Redraw the current display with fit and the ERROR plot to FORTRAN unit 18 in a format compatible with the LXY21.

\F – Redraw the current fit function over the fitting range and place best fit parameters on the screen.

\H – Closes current file (unit 18) and sets up a new output file tailored to a specific device or language. Waits for second character entry. Second character can be:

- E – Talaris 810.
- H – HP_GL - QMS PS-1700
- L – LXY21.
- Q – QUIC - Talaris 1590.
- T – TEK 4014 - Talaris 1590 (Default).

\K – Restore all deleted points in latest 1-2 range.

\L – Set the LOWER bounds of the fitting region to the x-position of the current cursor. When the plotting region is changed, if part of the fitting region is excluded by the change, the fitting region is redefined to coincide with the plotting region as necessary.

\M – Change line profile model for fitting. Waits for second character entry. Second character can be:

- S – Simple gaussian model.
- I – Integrating gaussian model.
- G – Gaussian model plus step.
- N – Nominal double-gaussian line model.
- D – Double-gaussian line model with positronium continuum.
- 4 – 4 param double-gaussian model.
- 5 – 5 param double-gaussian model.
- 6 – 6 param double-gaussian model.
- K – K4 Param double-gaussian model (no step, no ratio).
- L – L5 Param double-gaussian model (no step).
- ? – Lists available line models.

\N – Exit (same as N but sets flag in COMMUNICATIONS common block)

\P – Encode the current cursor position in world (plot scale) coordinates and store in the specified element of the OTHERLABELS array. Command requires additional character inputs to identify axes to encode and OTHERLABELS index for storage. Valid forms for additional characters are:

<n> – Encode both x-axis and y-axis positions and store in OTHERLABELS array with index <n>. <n> represents a number from 0 thru 9.

X<n> – Encode only the x-axis position and store in OTHERLABELS array with index <n>.

Y<n> – Encode only the y-axis position and store in OTHERLABELS array with index <n>.

The encoded positions can then be placed on the display using the \A<n> cursor command.

\R – Redraw the current plot to a file pointed to by logical IGLPIC (FORTRAN logical unit=18 and default file name=IGLPIC.DAT). Output to this file is in a format compatible to a specified output device or language. See \H (default = TEK 4014). If a fit has been done since the most recent change of region or Z command, the fit and fit parameters are included as well.

\U – Set the UPPER bounds of the fitting region to the x-position of the current cursor. When the plotting region is changed, if part of the fitting region is excluded by the change, the fitting region is redefined to coincide with the plotting region as necessary.

\V – Draw an arrow from position 1 to position 2. Command takes a second character input. The second character may be:

1 – Place arrowhead at 1 end of arrow.

2 – Place arrowhead at 2 end of arrow.

0 – No arrowhead.

A maximum of 50 arrows may be assigned.

\W – Delete the arrow the cursor is resting on. (No action if cursor is not resting on an arrow.) If two arrows are superimposed, only one is deleted.

\X – Extrapolate and draw on graph current fit function to cover region defined by 1 and 2 x-positions.

Index