0. XAS Programmer Guide Table of Content

Preliminaries

Textual help pages

1. XAS Programming Help top menu

Introduction to be written. See table of content of existing document.

2. The XAS directory tree

The XAS installation comes as a directory tree rooted under a logical position referred to by environment variable `$XASTOP`. It shall not contain any file (except for makefiles created by xasbuild or other compilation support files, but only the subdirectories listed below.
This does not refer to the area where the data is stored, which is instead arranged freely by each individual users according to the guidelines described elsewhere

The subdirectories are

- The `bin` directory
- The `calib` directory
- The `config` directory
- The `demo` directory and whatever other additional source directories may be provided with contributed or user software (they are all functionally equivalent to the `source` directory and are therefore not described

in any further detail).

[Previous][Next] [Up][Down]

2.1 The `bin` directory

The `$XASTOP/bin` directory is the target directory of build, i.e. the place where all the binary executables of XAS commands (main programs) will reside.

As such it is the only directory which will need to go in each user's path.
Note that all "official" XAS commands are binary executables, there are no such thing as officially supported XAS (shell) scripts because of the original architectural choice of XAS to be system-independent (at the time it was designed that meant running on VAX VMS and several Unix flavours), i.e. not dependent of any shell.

As such this directory will contain system-dependent files. A multi-architecture support of XAS will require separate `bin` directories under different `$XASTOP`s

Note that because of the above architectural choice, some XAS commands are implemented as front end wrappers which chain to another executable (i.e. they run for a while then are overlaid by another executable). This will be specified in the program specific manual page.

[Previous][Next] [Up][Down]

2.2 The `calib` directory

The `$XASTOP/calib` directory is not actually a software directory like all others in the typical XAS tree, but is more a data directory.

It is intended to contain calibration data files, which are described elsewhere.
The software will look (at open time, for files designated in the `CALIB` category) in the appropriate subdirectory of `$XASTOP/calib` for such files, although one can override the standard calibration files with private copies using the mycaldir XAS variable to point to an alternate location.

The directory has the following subdirectory arrangement :

- `general`
  currently empty, intended for spacecraft independent data
- one directory for any other spacecraft mission (this can be accessed setting the spacecraft XAS variable. The only one officially supported so far is
- `sax`
  For calibration data related to the BeppoSAX mission, arranged in subdirectories
  - `sc`
    contains descriptive files on spacecraft telemetry packet and HK parameters
  - `lecs`
    contains unofficial files for the LECS instrument (both descriptive of telemetry and HK and instrument calibration data)
  - `mecs`
    contains files for the MECS instrument (both descriptive of telemetry and HK and full instrument calibration data)
  - `hpgs`
    contains files for the HPGSPC instrument (both descriptive of telemetry and HK and a few instrument calibration data)
  - `pds`
    contains files for the PDS instrument (both descriptive of telemetry and HK and the needed instrument calibration data)
  - `wfc`
    is reserved for files for the WFC instruments (currently unused)

[Previous][Next] [Up][Down]

2.3 The `config` directory

The `$XASTOP/config` directory is not actually a software directory like all others in the typical XAS tree, but a support directory for the xasbuild software tool used for the building (compilation or recompilation) of XAS.

The directory has the following subdirectory arrangement (the functions are described elsewhere) :

- `Depend`
  contains the dependency files for all main programs, all libraries and (subdirectories for) the frozen versions (all this stuff is generated by xasbuild)
- `Lists`
  contains the list files listing components like subroutines in a library, main programs in a source directory, supported source directories, etc. (plus subdirectories for the frozen versions). This stuff is updated and mantained by xasbuild from a minimal bootstrap version.
- `Log`
  contains the log files of the xasbuild runs (can be cleared at user's will)
- `Proto`
  contains the prototypes (plus subdirectories for the frozen versions) of the Makefiles which xasbuild will use to install the system dependent ones in the (program and library) source code directories. These are the most important bootstrap files in the distribution.
- `Scripts`
  contains the shell scripts which constitute the xasbuild software package

[Previous][Next] [Up][Down]

2.4 The `doc` directory

The `$XASTOP/doc` directory contains a few (ASCII) documentation files :

- the release notes for the major "releases" of XAS (these have names like `xas11.relnotes` for release 1.1)
- the XAS porting note `port.txt` to be read by those intending to port XAS to another operating system
- the buglist notes for the major "releases" of XAS (these have names like `xas20.buglist` for release 2.0)

Note that the concept of releases is quite relaxed. XAS was not intended to have formal releases, but to be updated dynamically, with users picking up updated files and integrating them in their own version with xasbuild. After an early attempt at TESRE, this was not supported by SDC. At the moment there is no longer any active support to XAS at SDC,

[Previous][Next] [Up][Down]

2.5 The `external` directory

The `$XASTOP/external` directory contains the source files of library subroutines which were not developed as part of XAS, but were procured externally. The resulting relocatable libraries must be generated in the same place as XAS libraries, but this will not be done using xasbuild (it is either done by tools provided with the library or has to be done by the user).

Current external libraries are the following (it shall be noted that each of them is used only by a very small number of XAS programs) ; there is a subdirectory per library.

- `fitsio`
  Fortran sources for FITSIO 4.14. FITSIO routines are used only by the fromfits, tofits, fromogip, toogip command. They were compiled and tested with version 4.14. Usage with later release at user's risk.
- `ssdaux`
  Fortran and C routines provided by SSD (with a front-end wrapper by Daniele Dal Fiume) used mainly by the saxauxcalc command to compute auxiliary quantities (function of orbit and attitude data).
- `pgplot`
  currently empty, preserved for historical reasons (PGPLOT was used in some early graphics demos, and replaced by a dummy wrapper on non-VMS systems). No longer in use.

[Previous][Next] [Up][Down]

2.6 The `include` directory

The `$XASTOP/include` directory contains the Fortran `INCLUDE` files used by main programs and subroutines (and managed by xasbuild at compile time), plus some other support files read by programs (since they are not properly data files they are located here and not with calibration files.

- Fortran include files have all extensions `.inc`
- Files of type `.list` are miscellaneous support files like error code lists, font lists or graphics marker lists.
- Files of type `.prologue` are miscellaneous support files , and namely are Postscript prologues used by the Postscript-based graphics servers.

[Previous][Next] [Up][Down]

2.7 The `lib` directory

The `$XASTOP/lib` directory is a target directory of build, i.e. the place where all the relocatable libraries will reside.

As such this directory will contain system-dependent files. A multi-architecture support of XAS will require separate `lib` directories under different `$XASTOP`s

The following is a list of XAS libraries (external libraries are also present here, and other libraries used by unofficially contributed programs can also be placed in the `lib` directory) The respective routines in source form are located in `libsource` or `vos`. The name of the relocatable library file depends on the operating system (e.g. `lib`*`name`*`.a` under Unix or *`name`*`.OLB` under VMS).

library name	purpose
fotlib	routines specifically used to access SAX telemetry on Final Observation Tapes
general	general purpose routines (e.g time-and-date, byteswap etc.)
graphserv	low level library for the graphics servers, and the relevant clients
hpgslib	routines specific of the SAX HPGSPC instrument
lecslib	routines specific of the SAX LECS instrument (unofficially supported)
mecslib	routines specific of the SAX MECS instrument
pdslib	routines specific of the SAX PDS instrument
vos	system dependent routines
xasgraph	high level graphics library
xaslib	routines independent of SAX data formats, but specific to handle mission-independent XAS data files and features

[Previous][Next] [Up][Down]

2.8 The `libsource` directory

The `$XASTOP/libsource` directory is a source directory of build and is system-independent.

It contains a directory per library, and each directory contains Fortran source files (one per routine, except for service routines called only by other routines, which are included in the same source file as the caller), or exceptionally C source files (two cases).

Individual routines are listed in a separate section. The following is a list of XAS libraries (external libraries are described elsewhere). The system dependent `vos` library has sources elsewhere.

library name	purpose
fotlib	routines specifically used to access SAX telemetry on Final Observation Tapes
general	general purpose routines (e.g time-and-date, byteswap etc.)
graphserv	low level library for the graphics servers, and the relevant clients
hpgslib	routines specific of the SAX HPGSPC instrument
lecslib	routines specific of the SAX LECS instrument (unofficially supported)
mecslib	routines specific of the SAX MECS instrument
pdslib	routines specific of the SAX PDS instrument

xasgraph	high level graphics library
xaslib	routines independent of SAX data formats, but specific to handle mission-independent XAS data files and features

[Previous][Next] [Up][Down]

2.9 The `local` directory

The `$XASTOP/local` directory is intended as a repository for site-dependent support files which can be customized at each site or machine.

At the moment there a single file in this directory, `tape.cmds`, which is used to customize the system tape handling commands used by the fotfile program (actually by the system script written by fotfile). A second template `tape.cmds_remote` is provided as example for the case one wishes to access a remote type drive via rsh. The relevant file is described together with other miscellaneous support files

[Previous][Next] [Up][Down]

2.10 The `source` directory

The `$XASTOP/source` directory is a source directory of build and is system-independent.

It contains Fortran source files (one per XAS command main program, eventually inclusive of any routine called exclusively by such program, or by any routine overriding a library routine of same name), or exceptionally C source files (one case).

Details of individual programs are described elsewhere

[Previous][Next] [Up][Down]

2.11 The `vos` directory

The `$XASTOP/vos` directory is a source directory of build and is the only one to contain **system-dependent** code.

It is a single directory (which exists in many different versions for each supported system, of which only the one corresponding to your architecture shall be installed in `$XASTOP/vos`), which contains Fortran source files (one per routine, except for service routines called only by other routines, which are included in the same source file as the caller), and eventually C source files.

Individual routines are listed in a separate section.

VOS high level routines are Fortran-callable routines, and generally are actually Fortran routines with a very few exceptions.
Under VMS most of them call directly system libraries or services (with the exception of the memory allocation interface which uses a couple of C jacket routines).
Under Unix they call a tier of lower level C jacket routines (Fortran-callable) which in turn call C system routines (which may not be callable directly by Fortran, usually because of the underscore loader convention).

While the majority of VOS (Virtual Operating System) Unix routines are the same for all Unix flavours, there are a few routines supplied in different forms for the various supported Unix flavours.

[Previous][Next] [Up][Down]

3. How to build XAS

XAS executables are compiled and linked from source code using the `xasbuild` command, which generates appropriate Makefiles and runs them. This applies only to the Unix version, and is in principle capable of dealing with XAS updates as well as with XAS module addition by the user.

The VMS version of XAS has no general build utility, only a bootstrap script useful for rebuilding the entire distribution from scratch could be supplied.

[Previous][Next] [Up][Down]

4. How to write a XAS program

This section is intended to give an overview of XAS idioms used in writing typical XAS programs.

Details specific of individual programs or calling syntax of individual routines are supplied elsewhere, as well as specifics of graphic programs.

- typical user interface
- accessing environment
- dealing with errors
- interrupting loops
- usage of dynamic memory
- chaining more programs
- writing a dispatcher
- opening typical files (seq and direct)
- dealing with XAS image files
- dealing with XAS tabular files
- dealing with XAS headers
- dealing with data conversion
- dealing with times
- a SAX accumulator
- pipe communication (randomizer)
- using bit images
- pick some from list in section 10

[Previous][Next] [Up][Down]

5. XAS libraries

This section provides an index of XAS library routines in alphabetic order and by subject, which point to the appropriate syntax description in the relevant library section (XAS libraries are listed also in section 2.7).

Note that in the following sections a consistent notation and color code is used for routine synopsis, with a table entry like the following :

- **routine name**

Library	library name	pointer to Fortran or C code
Calling sequence	`CALL routine(arg1,arg2,arg3)`	
Arguments	Fortran type	ARG1 for intent=input
	Fortran type	ARG2 for intent=output (returned values)
	Fortran type	ARG3 for intent=inout (modified values)

Library list

library name	purpose
fotlib	routines specifically used to access SAX telemetry on Final Observation Tapes
general	general purpose routines (e.g time-and-date, byteswap etc.)
graphserv	low level library for the graphics servers, and the relevant clients
hpgslib	routines specific of the SAX HPGSPC instrument
lecslib	routines specific of the SAX LECS instrument (unofficially supported)
mecslib	routines specific of the SAX MECS instrument
pdslib	routines specific of the SAX PDS instrument
vos	system dependent routines

xasgraph	high level graphics library
xaslib	routines independent of SAX data formats, but specific to handle mission-independent XAS data files and features

---

Routine alphabetic list

- quick search from initial letter : ABCDE FGHIJ KLMNO P Q RST UVWXY Z

Routine subject list

You can locate a routine by subject using the following lookup table. A routine may eventually appear in more than one category. Routines in a category are grouped by further subject, and finally in alphabetic order.

Routines which are similar, or are called exclusively one by another, are listed side by side.

General	[files] [type conversion] [string manipulation] [time] [user i/f] [environment] [miscellanea] [initialization]
System	[memory] [pipes] [process control] [misc high level VOS] [misc low level VOS] [environment]
XAS file format	[files] [binary tables] [keywords]
XAS miscellanea	[time window] [attitude]
Graphics	[server communication] [high level] [low level]
SAX specific stuff	[FOT tape] [FOT telemetry] [support] [accumulation]
SAX instruments	
MECS	[calibration parameters] [matrix support] [accumulation]
LECS	[everything together] NB : unofficially supported
PDS	[everything together] NB : code by ITESRE, no full support by the author
HPGSPC	[everything together] NB : most code by IFCAI, no full support by the author ; rest of code unofficially supported

general file access

These routines shall be used to deal with generic files (in replacement of the standard Fortran `OPEN` and `INQUIRE` calls). Note that for the majority of data files one uses the XAS file routines (which in turn call these as underlying layer).

file opening	assign logical unit	free_lu
	convert sys to VOS filename	z_sys_name
	open generic file	z_open
	open ASCII table	open_xas_ascii
error handling	VOS errors	voserror
	Fortran errors	z_syserror
file status	(INQUIRE)	z_inquire zc_stat
file handling	delete	z_delete_file zc_unlink
	rename	z_rename_file zc_rename

Data type conversion support

XAS data is kept in native binary format. However some programs must (or are capable to) deal with data in foreign binary format. These routines are used to assist in the conversion.

preliminaries	operating system identification	z_op_sys blksyscommon
	identify required conversion	conversion_needed
	IEEE to VMS and v.v.	z_ieee_to_vms z_vms_to_ieee z_dieee_to_vms z_dvms_to_ieee
	byte swapping (little-big endian and v.v.)	swapi2 swapi4 swapr8

conversion		
miscellanea	unsigned DOUBLE PRECISION	udouble
	hex formatting	hexi4

General string manipulation

string length without trailing blanks	true_length
remove multiple blanks	collapse
smart replacement for `INDEX`	bindex
Case conversion	lowcase upcase
unsupported ?	trimroot

General date and time handling

time conversion	time array to Unix	time_1970
	Unix to time array	time_array
	Unix to string	time_ascii
System times	current time	zc_time
	time of login	z_logintime
Unsupported	HPGSPC routines	time_70s2mjd time_a2mjd time_cldj

User interface

These routines are used to enforce the XAS user interface reading from standard input, run string or command file. They shall be used as replacement for the `WRITE(*,'prompt')` and `READ(*,*)` Fortran idioms.

Basic routines	issue prompt	x_prompt
	read reply	x_read
	parse string arrays	preparse
Low level routines	retrieve and parse run string	z_get_command zx_get_parameter blkxrcommon
Unsupported	PDS routines	x_echo x_echo_error

Environment access

These routines are used to access XAS environment variables. The technical implementation of such variables is system dependent :

- on VAX VMS uses naturally DCL global symbols prefixed with `XAS_`
- on Unix uses environment variables prefixed with `XAS_` but saves them to a disk file to allow inheritance from child to parent processes.

XAS environment variables are accessed without the `XAS_` prefix. The system, searching for *var* looks first for `XAS_`*var*, but falls back to the system variable *var* if not found (system variables are however readonly for what XAS programs are concerned).

Get variable value	get_global_default z_get_global zc_getenv
Set variable value	z_set_global zc_putenv
Service routines (Unix)	z_aux_envfile z_initenv

General miscellanea

CURFIT fitting package	curft fchi fder mtinv
interpolation	extrp extrpd interpolate
matrix/vector operations	dotproduct matproduct
Random number generator	ran1
Bit array handling	bit_init_handle

Block data initialization

These `BLOCK DATA` routines are used/referred to initialize some specific common blocks, described in the include file section.

Binary table descriptors	blkbincommon
Current context	blkctxcommon
XAS file buffers	blkhcommon

for communication channels	blkpipecommon
For data conversion	blksyscommon
For user interface	blkxrcommon

VOS memory allocation

routines are used to allow dynamic memory allocation in f77.	z_alloc z_dealloc zc_alloc zc_free

VOS communication channel support

Communication channels are used to allow interprocess communication, typically for graphics servers. The implementation is system dependent :

- on VAX VMS uses mailboxes and Fortran unformatted sequential i/o
- on Unix uses named pipes, but differs according to the flavour.
- on Ultrix and Digital Unix uses Fortran unformatted sequential i/o
- on other Unixes uses a jacket routine to C i/o

create/open/delete channel	z_channel zc_mknod z_open_stream zc_fopen
close channel	z_close_stream zc_fclose
input/output routines	z_read_stream zc_fread z_write_stream zc_fwrite
unsupported	z_seek_stream zc_fseek

VOS process control

These routines are used either to control the current process, or to generate a new process.

terminate (replaces Fortran STOP)	z_exit
Handle interrupt (control-C)	z_break zc_break
Chain another process (overlay)	z_run zc_execvp
Schedule process in background (no wait)	z_spawn zc_fork
Schedule process with wait	z_schedule zc_system

VOS misc high level calls

The VOS library groups all system dependent calls. Those which are not listed elsewhere are listed here, limiting however to the routines which are called by user programs directly.

Error handling		voserror
Query functions	user name (GECOS and account)	z_fullname z_username
	hostname	z_hostname
	time of login	z_logintime
	terminal name and characteristics	z_terminal z_ttyname
Debugging utilities		checkerr cpuclk
Unsupported	(used by non XAS programs)	z_print_file z_tape_open

VOS misc low level calls

These are all C routines and all (with one exception) required on Unix only as underlying layer to the VOS high level calls.

support to query calls	zc_cuserid zc_getdomainname zc_gethostname zc_pwnam zc_terminal zc_ttyname
miscellanea	zc_dtime zc_memcpy

XAS file support

XAS (mission-independent) format data files are accessed via dedicated routines. We list here the general or generic ones, leaving out those related to tabular files and those used for the (common) XAS file header

Path related routines	locate path for file	buildpath
	strip path from filename	depath depath_1
generic XAS file	create/open	open_new_xas_file open_old_xas_file
	close	close_xas_file
image format files	generic images	create_image open_image read_image
	Response matrices	open_matrix xasmatout

XAS binary table support

Spectra, time profiles and photon lists are XAS files sharing a common tabular format. The related routines are listed here.

file opening	creation of new file	create_photon create_spectrum create_time
	existing file	open_photon open_spectrum open_time
read/write	a generic record	read_bin write_bin
preliminaries	handling table descriptors	copy_table_desc get_table_desc set_table_desc
	space for dummy columns	pad_table
service routines	for FITS-like TFORM keywords	decodetform leftnumber maketform

XAS file header and keyword handling

All XAS file share a common header format, composed of named binary keywords. There are high level routines to manipulate keywords, and low level service ones.

user routines	add and format HISTORY keyword	addhistory
	add/modify keyword	h_add_keyword h_modify_keyword
	read keyword value	h_read_keyword
	copy entire header	h_copy_all_header
	if file size changed	h_update_datasize
service routines	point to different file	h_current_file
	load header from disk	h_load_header h_load_minih
	flush header to disk	h_flush_header h_flush_minih
	Keyword seek support	h_find_keyword h_next_keyword

Accumulation : time window management

XAS accumulation program are mission dependent (although this is hidden under a mission-independent interface). They however use a common (mission-independent) concept of time windows, therefore the relevant routines are listed here.

prepare to open time window file	init_timewindow
open time window file	openwindow
read time window file	loadwindow

Attitude handling

These routines are designed in a mission-independent way, but are used only by unofficial programs using celestial coordinates

Determine pointing	get detector pointing	detpointing
	get spacecraft pointing	satpointing
	get/apply misalignments	getmisalign misalign
	Eulker matrix to sky coordinates	radecroll
Coordinate conversion	Pixel to mm and v.v.	lintomm unlintomm mmtopix
	Sky to pixel and v.v.	skytoxy xytosky gnomonic

Graphics server (communication)

A graphics server has to be created, and communication has to be set up over a communication channel from server and client sides.

Connection	establishment	connectserver
	verify channel existence	pipeexist
	server registration	isregistered register deregister
Xlib interface	for X window server	f2x

High level graphics

This set of routines correspond to relatively complex graphics functions implemented in graphics clients.

plot annotations	(two styles)	annotate_new annotate_old

Default arrangements	axis frame with annotations	df_axes
	pen colours	df_pen_colours
	plotting window and viewport	df_viewport df_window
Setup from environment	clear screen on overtrace	check_overtrace
	plot style (solid, error bars etc.)	get_datastyle
Axis frames	labelling routines	lb_axis lb_number lb_tics
	various axis styles	nice_axes nicer_lin_axes nicer_log_axes
Data array plotting	various forms	plot_xxy_bar plot_xxy_histo plot_xxy_join plot_xy_join

Low level graphics

These routines are implemented at device independent level and send standard opcodes (with operands) on a communication channel to a graphics server (and eventually receive replies).

Establish communication	(or close)	y_openplot y_closeplot
set up	viewport and window	y_clear_viewport y_viewport y_window y_page
	reference frame	y_coordinates y_scale
	look and style	y_colour y_text y_width
Plotting primitives	vector	y_draw y_move y_lines
	text	y_text
	image and 2-d	y_fill y_readlut y_write_image y_writelut
	graphics input	y_get_cursor y_readlut

FOT tape access

There is no intrinsic (system and device-dependent) tape handling in XAS. SAX FOT tapes are dealt with via system-specific shell scripts which are generated based on a template file (which can be customized locally) which contains template commands which are adjusted by the following routines.

service functions	load template	tapechar
	begin and end operations	add_init add_end
	edit template command	edit_cmd
handle specific commands	file copy	add_file
	tape functions	add_rew_tape add_skip_tape

FOT (telemetry) data reading

SAX data is in form of (FOT reformatted) telemetry data files, or other auxiliary files assimilated to them. This family of low level routines deals with telemetry records (or events in a record) themselves, and are called at a bottom of a sequence of higher level routines by the accumulation programs.
The low level routine may call a front end correction routine (which in turn calls further instrument specific corrections), and do call an instrument-independent "increment routine" which depends on the XAS data type being accumulated.

basic type switching	1 direct mode data (events) 2 indirect (spectra) 3 Housekeeping	sax_acc_bt_1 sax_acc_bt_2 sax_acc_bt_3
specific type handling	1 direct event handling	sax_acc_b1s1_i sax_acc_b1s1_y sax_acc_b1s2_i sax_acc_b1s2_y sax_acc_b1s3_y
	exposure computation	exposure_b1s1 exposure_b1s3 timebin_b1s1
	instrument dep. event correction	correct
	2 indirect	sax_acc_b2s1_y
	3 Housekeeping	sax_acc_b3s1 sax_acc_b3s2 sax_acc_b3s3 sax_acc_b3s4 sax_acc_b3s5 sax_acc_b3s6
instrument directory	record decoding	rearrange_instrec

Telemetry data reading support

Accumulation program are mission specific, but try to be as mission-independent and instrument-dependent as possible using external packetcap files to describe the content of telemetry packets and dispatching to the above low level routines according to such description.

which packets	are available ?	check_packet

load packet info	mission-independent i/f	pktcap_load pktcap_lookup
	SAX specific i/f	sax_pktcap_load
HK parameter info	similar to above	sax_pcf_load sax_pcf_lookup
Open files	instrument telemetry	sax_acc_open_tlm
	spacecraft telemetry	sax_acc_open_sc_tlm
	directory information	sax_open_dir
dispatcher to	packet specific routines	sax_acc_loop

Accumulation program setup (user dialogue)

The accumulation programs must ask the user about the choice of limits for the accumulation, based on the content and layout of telemetry packets, and also perform other initializations.

operator dialogue	choose packet type	sax_which_data
	load packetcap information	sax_acc_preload
	select fields and ranges	sax_acc_select sax_acc_hkrange sax_acc_other_range sax_acc_range
	time profile binning	askbin asktime
initializations	spacecraft to XAS time	time_constants_setup get_start_end update_start_end
	obaervation chain	get_obs_chain
	instrument dep. corrections	init_correct
	XAS keywords in output	instr_keywords sax_df_keywords

MECS calibration parameters

SAX MECS specific routines are presented in a detailed breakdown. A first class is represented by the routines accessing calibration parameters, used mainly for response matrix generation, but not only (e.g. they are also used for the WWW MECS guided tour)

Mirror system	area_mr psf_mir psf_rad
Windows and filter incl. materials	abs_co_be bewin_trasp alum kapton2 lexan poly poly_carbo
Detector components	coda cross_sec escape gas_cell spread

MECS matrix generation support

The MECS matrix generation programs (besides the above "physical" routines) includes some specific service routines.

parameter choices	blsel ein eout
service computations	mecs_arf mecs_rmf multiply_rmfarf tetafi_xy
Output to XAS or OGIP	fopen_rmf mecs_matkeywords write_arf write_rmf write_rmf_ebo write_rmf_mat

MECS data accumulation

Data accumulation programs call instrument specific routines for event corrections, or to add instrument specific keywords to the output files.

corrections	init_correct_me init_correct_me_fast me_gain_time me_init_map mecscorrect
keywords	mecs_keywords

PDS support

PDS support routines have been written at ITESRE and are unsupported by the author of this document. No detail is given so far.

pds_arf pds_ein pds_en_resol pds_matinfo pds_matkeywords pds_matout pds_ogip pds_opnrmf pds_response pds_wrtrmf pds_wrtrmfebo pds_wrtrmfmat pdsmat_coef pdsmat_init pds_freq
init_correct_pds pdscorrect
inst_key_copy inst_key_find inst_key_flush inst_key_load inst_key_mult inst_key_read inst_key_set instrument_keys no_keyword pds_keywords
pds_fotunits rminmax sax_froot_name xdofit

LECS support

LECS support is unofficial. The routines below were either been written by the author mimicking the MECS arrangement, or were adapted from SSD routines.

adapted from SSD	blrng e2fwhm
MECS-style corrections	lecscorrect init_correct_le le_gain_time lecs_keywords

HPGSPC support

HPGSPC is partially unofficial supported by the author, mimicking the MECS arrangement. However most of the routines here were written at IFCAI for the original release of HPGSPC software (and their status of support is unknown).

supplied by IFCAI	broad2 cofas fuga fuga_1 shell_fact shell_prob winbe fopen_rmf reader reader_1 write_arf write_rmf_ebo write_rmf_mat effmed filecorr_read fileinp_read parameter tmed tofits buf_read config_read julia togdp
MECS-style corrections	init_correct_hp hpcorrect hp_gain_time hp_keywords

[Previous][Next] [Up][Down]

5.1 The vos library

The `vos` library groups system-dependent routines (the Virtual Operating System) whose top layer has a standard system-independent calling sequence. There is a variant of each routine at least for Unix and VMS, and often for each flavour of Unix. The routines in the bottom layer (when existing) can be radically different.

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

Note that the `zc_*` routines are C jacket routines in the lower layer, and will generally not exist for VMS. The code pointers will indicate the different versions for the various operating systems :

- **Unix** indicates a common version for all Unix and Linux flavours
- **Linux** indicates something specific of Linux (tested with the Intel compiler)
- **DEC** indicates a common version for Ultrix and Alpha (OSF1 aka DU aka Tru64)
- **Ultrix** indicates the Ultrix version
- **OSF** indicates the Alpha (OSF1 aka DU aka Tru64) version
- **Sun** indicates a common version for SunOS and Solaris on Suns
- **SunOS** indicates SunOS specific routines
- **Solaris** indicates Solaris specific routines
- **HP-UX** is supported via code produced at ITESRE, but not available to the author of this document, and therefore is not listed here (where different from the common Unix set)
- **VMS** indicates code tested and used under an old release of VAX VMS (there is no support to OpenVMS)

The Linux version has been (mostly) set up in 2005 by Giorgio Calderone and Luciano Nicastro at IASF Palermo in a different build arrangement (they tried to set up a VOS generated by cpp-like directives from a source common to all systems), and tested under the Intel compiler. The VOS (with sources different from OS to OS where required) presented here is based mainly on the result of their cpp-like preprocessor, has been finalised in 2009 and is also intended for use with the Intel compiler. The main differences with either their sources, or the typical Unix sources, concern respectively the routine `Z_LOGINTIME` (which they did not test and required a Linux adhoc fix) and all the memory allocation routines, in which their version is followed, at variance with the common Unix version.

A-Y	blkxrcommon	checkerr	cpuclk		
Z_A-Z_C	z_alloc	z_aux_envfile	z_break	z_channel	z_close_stream
Z_D-Z_E	z_dealloc	z_delete_file	z_dieee_to_vms	z_dvms_to_ieee	z_exit
Z_F-Z_H	z_fullname	z_get_command	z_get_global	z_hostname	
Z_I-Z_N	z_ieee_to_vms	z_initenv	z_inquire	z_logintime	
Z_O-Z_Q	z_op_sys	z_open	z_open_stream	z_print_file	
Z_R-Z_S	z_read_stream	z_rename_file	z_run	z_schedule	z_seek_stream
	z_set_global	z_spawn	z_sys_name	z_syserror	
Z_T-Z_U	z_tape_open	z_terminal	z_ttyname	z_username	
Z_V-Z_Z	z_vms_to_ieee	z_write_stream			
Zc_A-Zc_E	zc_alloc	zc_break	zc_cuserid	zc_dtime	zc_execvp
Zc_F	zc_fclose	zc_fopen	zc_fork	zc_fread	
	zc_free	zc_fseek	zc_fwrite		
Zc_G-Zc_N	zc_getdomainname	zc_getenv	zc_gethostname	zc_memcpy	zc_mknod
Zc_P-Zc_S	zc_putenv	zc_pwnam	zc_rename	zc_stat	zc_system
Zc_T-Zc_Z	zc_terminal	zc_time	zc_ttyname	zc_unlink	

this first family is made almost exclusively of Fortran code, and represents the layer of routines callable by the user (except where otherwise stated)

- ## blkxrcommon

Library	vos	Fortran code Unix [VMS]
Calling sequence	`EXTERNAL BLKXRCOMMON`	

This `BLOCK DATA` routine is implicitly called by x_read and defines in a COMMON block the information concerning the logical units associated to stdin, stdout and stderr, the largest logical unit number, and the FORMAT to be used for echoing a prompt.

- ## checkerr

Library	vos	C code Unix
Calling sequence	INTEGER ERRNO=CHECKERR()	

A debugging aid (unofficial) dto make the C `errno` available to Fortran programs.

- ## cpuclk

Library	vos	Fortran code Unix [VMS]
Calling sequence	`CALL CPUCLK('START')` `CALL CPUCLK('any text')`	

A debugging aid (unofficial) which prints the CPU time spent since last call i.e. during a phase of the program indicated by a label `'any text'` (the routine must be initialized once by a `START` call)

- ## z_alloc

Library	vos	Fortran code Unix [Linux] [VMS]
Calling sequence	`CALL Z_ALLOC(nelem,elsize,array,address,offset)`	
Arguments	`INTEGER`	`NELEM`
	`INTEGER`	`ELSIZE`
	`any`	`ARRAY`
	`INTEGER`	`ADDRESS`
	`INTEGER`	`OFFSET`

This routine is the **dynamic memory allocation** interface. One formally requests an extension of `NELEM` elements of size `ELSIZE` (in bytes, 1 for characters, 2 for INTEGER*2, 4 for INTEGER or REAL, 8 for DOUBLE PRECISION) to the `ARRAY` handle (it is usually enough to declare it ARRAY(1) in the caller). The routine **returns** the `ADDRESS` of the allocated area (which is unused except by the z_dealloc call, and the `OFFSET` (in units of `ELSIZE`) which allows to access the extension elements as `ARRAY(i+OFFSET)`.
On all operating systems it uses an underlying zc_alloc call to do the actual job.
Note that addresses are assumed to be 32-bit quantities, hence on 64-bit systems like Alpha OSF all code must be compiled with the `-taso` (Truncated Address Space Option).

See elsewhere for the idioms about usage of this routine.

- ## z_aux_envfile

Library	vos	Fortran code Unix
Calling sequence	`CALL Z_AUX_ENVFILE(FILE)`	
Arguments	`CHARACTER*(*)`	`FILE`

This routine is not intended for public use and must no be called. It is used on Unix by the z_get_global and z_set_global calls (actually by z_initenv) to build the name of a file used to save a copy of the XAS environment.
This is necessary only in Unix to allow back-inheritance (child to parent) of changes to the environment done by children processes.
The name of the file (residing in the user home directory) is of the form `ttyp`*`n_hostname`*`.environment`. The content of the file is not intended to survive a login session. If you wish to preserve the XAS environment of the last login session, do a `touch` of such file as first operation after login before issuing any XAS command.

- ## z_break

Library	vos	Fortran code Unix [VMS]
Calling sequence	`IF( Z_BREAK() )THEN ...`	

This logical function allows to detect if an interrupt of the current program has been requested (pressing the control-C key) and allows the program to jump to a dedicated piece of code (usually intended as a "gracious" way of interrupting prematurely a long loop).

The Unix version uses signal handling via zc_break while the VMS version handles an AST internally.

- ## z_channel

This routine is used to manage communication channels and has two entry points.

Library	vos	Fortran code Unix [VMS]
Calling sequence	`CALL Z_CHANNEL(pipename,myop)`	
Arguments	`CHARACTER*(*)`	`PIPENAME`
	`CHARACTER*(*)`	`MYOP ('CREATE'|'TEST'|'DELETE')`

The generic call above allows to perform on the named channel `PIPENAME` the operation specified by `MYOP` :

- `'CREATE'` : creation of the channel
- `'TEST'` : testing channel existence
- `'DELETE'` : deleting the channel

In Unix communication channels are named pipes on /tmp created by zc_mknod, tested and deleted as normal files (using zc_stat and z_delete_file).
In VMS communication channels are mailboxes in the job table created and tested via system services and automatically deleted when unused.

Calling sequence	`CALL Z_CHANNEL_OPEN(lu,pipename,'OPEN')`	
Arguments	`INTEGER`	`LU`

The open entry point above allows to perform on the named channel `PIPENAME` the opening operation `MYOP='OPEN'` which connects the channel to the Fortran logical unit `LU`.
For all operating systems opening is dealt with by z_open_stream

---

- ## z_close_stream

Library	vos	Fortran code Unix [Sun, HP-UX] [VMS]
Calling sequence	`CALL Z_CLOSE_STREAM(lu)`	
Arguments	`INTEGER`	`LU`

Closes a stream communication channel on logical unit `LU`. According to OS, it is either just a wrapper to a Fortran `CLOSE` or to the zc_fclose C jacket routine. See z_open_stream for details.

- ## z_dealloc

Library	vos	Fortran code Unix [Linux] [VMS]
Calling sequence	`CALL Z_DEALLOC(address)`	
Arguments	`INTEGER`	`ADDRESS`

Deallocates the memory block at `ADDRESS` allocated by z_alloc. Used seldom (memory is deallocated anyhow at exit) unless one wants to reallocate with a different size.
On all operating systems it uses an underlying call to zc_free do the actual job, except on Linux where it calls zc_alloc with the extra last argument set to 1.

- ## z_delete_file

Library	vos	Fortran code Unix [VMS]
Calling sequence	`Z_DELETE_FILE(vosname)`	
Arguments	`CHARACTER*(*)`	`VOSNAME`

Deletes a file with VOS file name `VOSNAME` using a call to zc_unlink(Unix) or a library function (VMS)

- ## z_dieee_to_vms

Library	vos	Fortran code Unix [VMS]
Calling sequence	`CALL Z_DIEEE_TO_VMS(DATA,N)`	
Arguments	`DOUBLE PRECISION`	`DATA(N)`
	`INTEGER`	`N`

Category: data type conversion
Converts an array `DATA` of `N` double precision values from IEEE format to VMS D-format. On Unix is not implemented (no-op), on VMS calls the NOAO assembler routines IEEUPD or IEEVUD.

- ## z_dvms_to_ieee

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_DVMS_TO_IEEE(DATA,N)	
Arguments	DOUBLE PRECISION	DATA(N)
	INTEGER	N

Category: data type conversion
Converts an array DATA of N double precision values from VMS D-format to IEEE format. On VMS is not implemented (no-op), on Unix is currently not implemented (but should be ! see ideas in the code !).

- ## z_exit

Library	vos	Fortran code Unix [Sun] [VMS]
Calling sequence	CALL Z_EXIT(RETCODE)	
Arguments	INTEGER	RETCODE

Replacement for Fortran statement STOP retcode, used to terminate a program passing back a return code or status code (use RETCODE=0 for normal termination, use a positive return code for errors, use a negative retcode only for graphics servers for which saving the environment to disk is not desired).

VMS version just handles appropriately the return code, while the Unix version actually passes control to (z_runs) an auxiliary C program savenv which saves the XAS environment to disk (via z_aux_envfile). There is a difference among Unix flavours concerning the fact whether logical units must be closed unconditionally or only if open, and whether stdin, stdout, stderr must be closed. On Sun this might interfere with i/o redirection.

- ## z_fullname

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_FULLNAME(USER,FULL)	
Arguments	CHARACTER*(*)	USER
	CHARACTER*(*)	FULL

Category: Query calls
This routine **returns** the full user name (GECOS field) FULL given the account username USER. Uses zc_pwnam in Unix and system services in VMS.

- ## z_get_command

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_GET_COMMAND(runstring)	
Arguments	CHARACTER*(*)	RUNSTRING

Category: User interface
Returns the entire RUNSTRING used to invoke the program. In Unix this is done by repeating IARGC() calls to GETARG and reassembling invidual arguments together. in VMS a single LIB$GETFOREIGN call suffices (the run string is then massaged a bit).

- ## z_get_global

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_GET_GLOBAL(NAME,VALUE)	
Arguments	CHARACTER*(*)	NAME
	CHARACTER*(*)	VALUE

Category: Environment access
Returns the VALUE of the XAS environment variable of given NAME (if missing, tries a system environment variable of same name, and if missing returns a blank string).
The VMS version uses global symbols, while the Unix version uses environment variables (however initialized from a disk saved environment via z_initenv

---

- ## z_hostname

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_HOSTNAME(HOST,DOMAIN)	
Arguments	CHARACTER*(*)	HOST
	CHARACTER*(*)	DOMAIN

Category: Query calls

This routine returns the current hostname HOST and also the Internet domain name DOMAIN. Uses zc_gethostname and zc_getdomainname in Unix while in VMS accesses UCX logicals or, if undefined, gets the SYS$NODE logical and assigns .decnet as domain.

• z_ieee_to_vms

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_IEEE_TO_VMS(DATA,N)	
Arguments	REAL	DATA(N)
	INTEGER	N

Category: data type conversion
Converts an array DATA of N REAL values from IEEE format to VMS format. On Unix is not implemented (no-op), on VMS calls the NOAO assembler routines IEEUPR or IEEVUR.

• z_initenv

Library	vos	Fortran code Unix
Calling sequence	CALL Z_INITENV	

This routine is not intended for public use and must no be called. It is used on Unix by the z_get_global and z_set_global calls. It builds the name of a file used to save a copy of the XAS environment via z_aux_envfile, then, if the file is older than the beginning of the current login session, deletes it and creates an empty one, otherwise it restores the current process environment from the list of variables stored in the disk file. The above operations are done only in the first call in a given program
This is necessary only in Unix to allow back-inheritance (child to parent) of changes to the environment done by children processes.

This routine contains **imbedded system-dependent code** to handle the different Fortran conventions about record lengths in file opening (this is the only routine which must use a Fortran OPEN instead of a call to z_open

---

• z_inquire

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_INQUIRE(vosname,query,return,creturn)	
Arguments	CHARACTER*(*)	VOSNAME
	CHARACTER*(*)	QUERY
	INTEGER or LOGICAL	RETURN
	CHARACTER*(*)	CRETURN

Replacement for the Fortran INQUIRE statement. Issues a query about a file with VOS file name VOSNAME and **returns** the appropriate value. Types of QUERY are :

- 'EXIST' tests file existence and returns a LOGICAL value in RETURN
- 'RECL' returns an INTEGER value in RETURN, the record length in bytes. For unopened file this information is unavailable in Unix, however for XAS files it can be derived from the mini-header, while for other files is emulated asking the user. The information is native in VMS.
- 'OPENED' tests if file is open and returns a LOGICAL value in RETURN
- 'PROTECT' returns in CRETURN the file protections in the form 'RWXRWXRWX' (user, group and other, ignore VMS system protection) or e.g. 'RWXR--R--' where a dash indicates a protection is unset.
- 'CDATE' returns the creation date in RETURN as an Unix time
- 'MDATE' returns the modification date in RETURN as an Unix time
- 'SIZE' returns an INTEGER value in RETURN, the file size in bytes

The Unix version uses in general the zc_stat call to get information, while the VMS version either "normalizes" the output of an INQUIRE statement or uses auxiliary RMS calls.

• z_logintime

Library	vos	Fortran code [Linux] [Alpha] [Ultrix] [SunOS] [Solaris] [VMS]
Calling sequence	CALL Z_LOGINTIME(TIME)	
Arguments	INTEGER	TIME

Category: Query calls
This routine **returns** the Unix TIME of the beginning of the current login session. This information for Unix is read in Fortran from /etc/utmp (however each Unix flavour has itw own peculiarities about such file), while for VMS is obtained via a system service.

• z_op_sys

Library	vos	Fortran code [Linux] [Alpha] [Ultrix] [SunOS] [Solaris] [VMS]

Calling sequence	`CALL Z_OP_SYS(system)`	
Arguments	`CHARACTER*3`	`SYSTEM`

Category: Query calls
This routine **returns** an hardwired three-character code `SYSTEM` identifying the operating system. By definition a different version of this routine is necessary for each operating system.

- ## z_open

Library	vos	Fortran code Unix [Sun] [VMS]
Calling sequence	`CALL Z_OPEN(lu,vosname,access,status,recl)`	
Arguments	`INTEGER`	`LU`
	`CHARACTER*(*)`	`VOSNAME`
	`CHARACTER`	`ACCESS ('Seq'|'Dir')`
	`CHARACTER*2`	`STATUS ('New'|'OLd'|'Unknown'|'OVerwrite',|'Append')`
	`INTEGER`	`RECL`

This routine is the replacement for the Fortran `OPEN` statement and associates a file with VOS file name `VOSNAME` with the logical unit `LU`.

Only two types of `ACCESS` are supported : sequential on formatted files and direct on binary files (in the latter case a record length (in byte for all systems) `RECL` must be supplied, use 0 for sequential files), specified by a (at least one-letter) code.

The opening `STATUS` is normalized as follows for all systems :

- an `'OLD'` file shall exist already
- a `'NEW'` file shall not exist already
- an `'OVERWRITE'` file will be deleted and recreated as new
- an `'UNKNOWN'` file covers any other case
- opening for `'APPEND'` is foreseen but not implemented

The routine is written entirely in Fortran (does not call any underlying routine) and takes care of system-dependent peculiarities, like : for Unix (except Sun) and VMS systems, the fact the record length in `OPEN` statements are in longwords (4 bytes) ; for DEC and VMS systems opening unwritable files in the allowed `READONLY` mode ; for VMS systems comparing the uer supplied record length with the one native of the system ; for VMS new sequential files, forcing them to `STREAM_LF` format.

- ## z_open_stream

Library	vos	Fortran code Unix [Sun, HP] [VMS]
Calling sequence	`CALL Z_OPEN_STREAM(lu,pipename,access,status)`	
Arguments	`INTEGER`	`LU`
	`CHARACTER*(*)`	`PIPENAME`
	`CHARACTER`	`ACCESS ('Binary'| 'Pipe'| 'Text')`
	`CHARACTER*2`	`STATUS ('New'| 'OLd'| 'Unknown'| 'OVerwrite',| 'Append')`

Opens a stream communication channel on logical unit `LU`. The channel name `PIPENAME` is passed by z_channel_open, which is the only publicly supported way to call this routine using `ACCESS='Pipe'` and `STATUS='OLd'`

The remaining values of `ACCESS` and `STATUS` (mimicked on z_open) are presently not supported by the basic version of this routine (like the Unix bases or VMS versions), which use plain Fortran sequential unformatted i/o (this is the only case in which this inherently unportable i/o is used, since no disk files are created (no exchange of data across machines), but just inter-process communication on the same machine.
The Sun and HP version, where Fortran sequential unformatted i/o cannot be used, is based on a wrapper zc_fopen to C stream i/o. This can in principle support also disk files but this is not used nor supported in any XAS program.

- ## z_print_file

Library	vos	Fortran code Unix [VMS]
Calling sequence	`CALL Z_PRINT_FILE(file)`	

This private routine is not part of XAS but has been used by other programs using the VOS library to programmatically print a file in a site dependent way.

- ## z_read_stream

Library	vos	Fortran code Unix [Sun, HP] [VMS]

Calling sequence	CALL Z_READ_STREAM(lu,buffer,recl)	
Arguments	INTEGER	LU
	CHARACTER*(RECL)	BUFFER
	INTEGER	RECL

This routine performs input on the communication channel opened on logical unit LU, namely reads a byte BUFFER of RECL bytes.
As explained for z_open_stream the basic implementation of channel input uses Fortran unformatted READ. On those Unix systems where this is not possible, C stream i/o via zc_fread is used.

- **z_rename_file**

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL SUBROUTINE Z_RENAME_FILE(vosnameold,vosnamenew)	
Arguments	CHARACTER*(*)	VOSNAMEOLD, VOSNAMENEW

This routine performs programmatically renaming of files using VOS file names VOSNAMEOLD and VOSNAMENEW. In Unix it calls zc_rename, and in VMS a system library call.

- **z_run**

A family of three routines dealing with process scheduling and control.

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_RUN(PROGRAMSTRING)	
Arguments	CHARACTER*(*)	PROGRAMSTRING

This routine overlays the current process with (passes control to) a new command (this implicilty terminates the current program, the new one continues in the same process space). PROGRAMSTRING can be another program or a shell script, followed by a list of arguments. Typically used by dispatchers, or multi-stage processes.
It is invoked in Unix via zc_execvp and in VMS via LIB$DO_COMMAND.

- **z_schedule**

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_SCHEDULE(PROGRAM,RETCODE)	
Arguments	CHARACTER*(*)	PROGRAMSTRING
	INTEGER	RETCODE

This routine schedules another process with wait (returns when finished with the status return code RETCODE of the child). PROGRAM can be another program or a shell script, followed by a list of arguments. Currently not used by any XAS program
It is invoked in Unix via zc_system and in VMS via a system process creation call.

- **z_spawn**

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_SPAWN(PROGRAM)	
Arguments	CHARACTER*(*)	PROGRAM

This routine schedules another process without wait (in background). PROGRAM can be another program (or perhaps a shell script), followed by a list of arguments. Typically used to start a (graphics) server.
It is invoked in Unix via zc_fork and in VMS via LIB$SPAWN.

- **z_seek_stream**

Library	vos	Fortran code Unix
Calling sequence	CALL Z_SEEK_STREAM(lu,n,recl)	

This routine is not supported nor used by any XAS program. It can be used in conjunction with the C variant of the communication channel stream i/o (explained with z_open_stream) to support random access to stream disk files (using zc_fseek).

- **z_set_global**

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_SET_GLOBAL(NAME,VALUE)	
Arguments	CHARACTER*(*)	NAME
	CHARACTER*(*)	VALUE

Category: Environment access
Sets the `VALUE` of the XAS environment variable of given `NAME` (use a blank `VALUE=' '` to effectively delete the variable).
The VMS version uses global symbols, while the Unix version uses environment variables (however initialized from a disk saved environment via z_initenv

---

- ## z_sys_name

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_SYS_NAME(VOSNAME,SYSNAME)	
Arguments	CHARACTER*(*)	VOSNAME
	CHARACTER*(*)	SYSNAME

This routine is not normally called publicly, but is called by any routine operating on files to convert a VOS file name `VOSNAME` to a system dependent filename `SYSNAME` to be passed to lower level calls.

A VOS file name is system independent, and assumes one of the following forms (most of them are Unix resemblant, but all work on all systems) :

- `~/dir/dir/file.typ`
  expanded as `$HOME/dir/dir/file.typ` (in VMS $HOME is equated to logical SYS$LOGIN)
- `$VARIABLE/dir/dir/file.typ` :
  the XAS or system environment variable is resolved by z_get_global before proceeding to further expansion
- `-/dir --/dir ---/dir`
  expanded as `../dir ../../dir ../../../dir`
- `/DIR/dir/dir/file.typ`
  interpreted as in Unix (in VMS this includes recognizing whether the first `/DIR` is a logical device name)
- `./dir/dir/dir/file.typ`
  interpreted as in Unix

- ## z_syserror

Library	vos	Fortran code Unix [VMS]
Calling sequence	IF (Z_SYSERROR (ierr,ivoserr,isyserr)) THEN ...	
Arguments	INTEGER	IERR
	INTEGER	IVOSERR
	INTEGER	ISYSERR

This logical function shall be called after each Fortran (i/o) statement which returns a status code `IERR`, and should take care to convert it to a standard VOS error code. VOS error codes can be looked at in the error code listings (but at present all Fortran errors are converted to a single VE_FORIOERR code).
It is used to branch to an error message or handler in case of error (returns `.TRUE.` if error occurred). It also **returns** the standard (VOS) error code `IVOSERR` and the corresponding system-dependent code `ISYSERR`.

- ## z_tape_open

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_TAPE_OPEN(TAPE,IERR)	

This private routine is not part of XAS but has been used by other programs using the VOS library to programmatically handle tapes in a site dependent way.

- ## z_terminal

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_TERMINAL(ROWS,COLUMNS)	
Arguments	INTEGER	ROWS,COLUMNS

Category: Query calls
This routine returns the number of `ROWS` and `COLUMNS` for the current terminal device. Uses zc_terminal in Unix and LIB$GETDVI in VMS.

- ## z_ttyname

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_TTYNAME(TTY)	
Arguments	CHARACTER*(*)	TTY

Category: Query calls
This routine returns the identifier of the current terminal `TTY`. Uses zc_ttyname in Unix (stripping the `/dev/` prefix) and LIB$GETJPI in VMS.

- ## z_username

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_USERNAME(USER)	
Arguments	CHARACTER*(*)	USER

Category: Query calls
This routine returns the current account name USER. Uses zc_cuserid in Unix and LIB$GETJPI in VMS.

- ## z_vms_to_ieee

Library	vos	Fortran code Unix [VMS]
Calling sequence	CALL Z_VMS_TO_IEEE(DATA,N)	
Arguments	REAL	DATA(N)
	INTEGER	N

Category: data type conversion
Converts an array DATA of N REAL values from VMS format to IEEE format. On VMS is not implemented (no-op), on Unix it does trivial byte handling. Note that this routine converts big endian IEEE floating point (i.e. Sun, HP-UX), for little endian IEEE (i.e. DEC) a previous call to swapi4 is necessary care of the caller.

- ## z_write_stream

Library	vos	Fortran code Unix [Sun, HP] [VMS]
Calling sequence	CALL Z_WRITE_STREAM(lu,buffer,recl)	
Arguments	INTEGER	LU
	CHARACTER*(RECL)	BUFFER
	INTEGER	RECL

This routine performs output on the communication channel opened on logical unit LU, namely writes a byte BUFFER of RECL bytes.
As explained for z_open_stream the basic implementation of channel input uses Fortran unformatted WRITE. On those Unix systems where this is not possible, C stream i/o via zc_fwrite is used.

this second family is made mainly of C code, and represents a layer of routines not intended to be called by the user (except where otherwise stated)

They are usually defined only for the various Unix flavours (VMS, except where stated, has routines directly callable by Fortran) as C jacket calls to system libraries (often not directly Fortran callable because of the underscore convention used by the loader, or inconveniently called because of awkward argument types).

- ## zc_alloc

Library	vos	C codeUnix [Linux] [VMS]
Calling sequence	IERR=ZC_ALLOC(nelem,elsize,array,address,offset [,mode])	

Back end of z_alloc, jacket to calloc. On Linux it is also back end to z_dealloc, jacket to free. This is supported by the extra argument mode (0 allocates, 1 deallocates), present only on Linux.

- ## zc_break

Library	vos	C codeUnix
Calling sequence	IERR=ZC_BREAK(1,Z_AUX_INTERRUPT_HANDLER)	

Back end of z_break, jacket to signal, enables the Z_AUX_INTERRUPT_HANDLER defined in z_break.

- ## zc_cuserid

Library	vos	C codeUnix
Calling sequence	IERR=ZC_CUSERID(name,length)	

Back end of z_username, jacket to cuserid.

- ## zc_dtime

Library	vos	C codeUnix [Ultrix, HP]

Calling sequence	DOUBLE TIME=DTIME(TIMES)	
Arguments	REAL	TIMES(2)

Back end of cpuclk, jacket to `times`.

- **zc_execvp**

Library	vos	C code Unix
Calling sequence	IERR=ZC_EXECVP(command,runstring)	

Back end of z_run, jacket to `execvp`, inclusive of extensive reparsing of the runstring parameters.

- **zc_fclose**

Library	vos	C code Unix [Sun,HP]
Calling sequence	unsupported	

Back end of z_close_stream, is dummy on systems using Fortran unformatted stream i/o, while the the Sun-HP variant used to support stream i/o in C is not officially supported.

- **zc_fopen**

Library	vos	C code Unix [Sun]
Calling sequence	unsupported	

Back end of z_open_stream, is dummy on systems using Fortran unformatted stream i/o, while the the Sun-HP variant used to support stream i/o in C is not officially supported.

- **zc_fork**

Library	vos	C code Unix
Calling sequence	IERR=ZC_FORK(command,runstring)	

Back end of z_spawn, jacket to `fork` and `execvp`, inclusive of extensive reparsing of the runstring parameters.

- **zc_fread**

Library	vos	C code Unix [Sun, HP]
Calling sequence	unsupported	

Back end of z_read_stream, is dummy on systems using Fortran unformatted stream i/o, while the the Sun-HP variant used to support stream i/o in C is not officially supported.

- **zc_free**

Library	vos	C code Unix [Linux] [VMS]
Calling sequence	IERR=ZC_FREE(address)	

Back end of z_dealloc, jacket to `free`. On Linux it is unused (untested ?) since `free` is jacketed in zc_alloc

- **zc_fseek**

Library	vos	C code Unix [Solaris]
Calling sequence	unsupported	

Back end of z_seek_stream, is not officially supported.

- **zc_fwrite**

Library	vos	C code Unix [Sun, HP]
Calling sequence	unsupported	

Back end of z_write_stream, is dummy on systems using Fortran unformatted stream i/o, while the the Sun-HP variant used to support stream i/o in C is not officially supported.

- **zc_getdomainname**

Library	vos	C code Unix
Calling sequence	IERR=ZC_GETDOMAINNAME(name,length)	

One of the back ends of z_hostname, jacket to getdomainname

- ## zc_getenv

Library	vos	C code Unix [Solaris]
Calling sequence	IERR=ZC_GETENV(variable,value)	

Back end of z_get_global, jacket to getenv

- ## zc_gethostname

Library	vos	C code Unix [Solaris]
Calling sequence	IERR=ZC_GETHOSTNAME(name,length)	

One of the back ends of z_hostname, jacket to gethostname (or uname for Solaris)

- ## zc_memcpy

Library	vos	C code Unix [VMS]
Calling sequence	CALL ZC_MEMCPY(BUFIN,BUFOUT,LOC,NBYTE)	
Arguments	any	BUFIN
	any	BUFOUT
	INTEGER	LOC
	INTEGER	MBYTE

Copies MBYTE bytes from the input area BUFIN to location BUFOUT(LOC:) (using a character notation). This routine is currently used for (also misaligned) memory copies by read_bin and by write_bin
It is a jacket to memcpy.

- ## zc_mknod

Library	vos	C code Unix
Calling sequence	IERR=ZC_MKNOD(path)	

Back end of z_channel, jacket to mknod

- ## zc_putenv

Library	vos	C code Unix
Calling sequence	IERR=ZC_PUTENV(variable,value)	

Back end of z_set_global, jacket to putenv

- ## zc_pwnam

Library	vos	C code Unix [Solaris]
Calling sequence		

Back end of z_fullname, jacket to getpwnam

- ## zc_rename

Library	vos	C code Unix [Solaris]
Calling sequence	IERR=ZC_RENAME(oldpath,newpath)	

Back end of z_rename_file, jacket to rename

- ## zc_stat

Library	vos	C code Unix [Sun,Ultrix]
Calling sequence	IERR = ZC_STAT(file,buffer)	

Back end of z_inquire (and others), jacket to stat

- ## zc_system

Library	vos	C codeUnix
Calling sequence	IERR=ZC_SYSTEM(command,runstring)	

Back end of z_schedule, jacket to `fork`, `execvp` and `signal`. The code has been derived by (and is closely resemblant to) a Sun listing of the `system` call, except that it runs a command directly, without any intermediate shell.

- ## zc_terminal

Library	vos	C codeUnix [Solaris]
Calling sequence	IERR=ZC_TERMINAL(ROWS,COLUMNS)	

Back end of z_terminal, jacket to `isatty` and an `ioctl`

- ## zc_time

Library	vos	C codeUnix [Solaris] [VMS]
Calling sequence	INTEGER TIME=ZC_TIME()	

Callable directly by Fortran, **returns** the current system `TIME` as an Unix time.
The Unix version is a C jacket to `time`, while the VMS version is written in Fortran and uses SYS$GETTIM and SYS$NUMTIM.

- ## zc_ttyname

Library	vos	C codeUnix [Solaris]
Calling sequence	IERR=ZC_TTYNAME(name,length)	

Back end of z_ttyname, jacket to `ttyname`

- ## zc_unlink

Library	vos	C codeUnix
Calling sequence	IERR=ZC_UNLINK(path)	

Back end of z_delete_file, jacket to `unlink`

[Previous][Next] [Up][Down]

5.2 The `general` library

The `general` library groups all (system-independent) routines which do not belong to any XAS, topic or instrument specific library.

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

A-C	bindex	bit_init_handle	blksyscommon	collapse	conversion_needed	curft
D-F	dotproduct	extrp	extrpd	fchi	fder	free_lu
G-M	gnomonic	hexi4	interpolate	lowcase	matproduct	mtinv
N-S	radecroll	ran1	swapi2	swapi4	swapr8	
T-Z	time_1970	time_array	time_ascii	true_length	udouble	upcase

- ## bindex

Library	general	Fortran code
Calling sequence	INTEGER FUNCTION BINDEX(all,part,i)	
Arguments	CHARACTER*(*)	ALL
	CHARACTER*(*)	PART
	INTEGER	I

This function is a replacement for `INDEX` which **returns** the absolute position of the n=th occurrence of string PART in string ALL, or better in substring ALL(I:), where the returned value I is used to keep context as in the following example which looks recursively for subsequent occurrences of the same string.

```
IS=BINDEX('pqrABCdefgABCxyz','ABC',1)           returns 4
PARTEND=IS+3-1                                  3 is the true length of PART
IS=BINDEX('pqrABCdefgABCxyz','ABC',PARTEND+1)   returns 11
```

The coding is just a trivial use of INDEX keeping account of the offsets.

- ## bit_init_handle

This LOGICAL function has 4 different entry points and is used to simulate a bit array (8 bits at a time are stored in a character array, thus NX*NY/8 bytes are used) using portable "character arithmetics" (ICHAR and CHAR functions).

Library	general	Fortran code
Calling sequence	LOGICAL dummy=BIT_INIT_HANDLE(nx,ny)	
Arguments	INTEGER	NX,NY

The initialization call allocates memory space for a bit array of dimensions NX times NY initialized to all zeroes.

Calling sequence	LOGICAL dummy=BIT_SET(ix,iy,value)	
Arguments	INTEGER	IX,IY
	INTEGER	VALUE [1|0]

The SET call sets (to 1, or resets to 0, according to VALUE the bit at position IX,IY.

Calling sequence	IF (BIT_GET(ix,iy) THEN ...	
Arguments	INTEGER	IX,IY

The GET call **returns** a .TRUE. value if the bit at position IX,IY is 1, and .FALSE. otherwise.

Calling sequence	LOGICAL dummy=BIT_NUMBER(ne)	
Arguments	INTEGER	N

while the NUMBER call **returns** the number N of bits in the array set to 1.

- ## blksyscommon

Library	general	Fortran code
Calling sequence	EXTERNAL BLKSYSCOMMON	

This BLOCK DATA routine is called implicitly by conversion_needed to initialize the SYSCOMMON common block used to keep track of the data conversion needed between a foreign operating system and the target (local) operating system. The initialization assumes no conversion needed.

- ## collapse

Library	general	Fortran code
Calling sequence	CALL COLLAPSE(string)	
Arguments	CHARACTER*(*)	STRING

This trivial routine collapses **in place** all duplicated blanks in STRING into a single blank. I.e. a string 'ABC__DEF' is returned as 'ABC_DEF'

- ## conversion_needed

Library	general	Fortran code
Calling sequence	CALL CONVERSION_NEEDED(local,foreign)	
Arguments	CHARACTER*(3)	LOCAL
	CHARACTER*(3)	FOREIGN

This call is used to fill the SYSCOMMON common block with the flags indicating specific data conversions are necessary (for integer, real, and character data respectively) between the FOREIGN operating system and the LOCAL operating system, where LOCAL and FOREIGN are the operating system codes returned by z_op_sys.

- ## curft

Library	general	Fortran code

Calling sequence	See Bevington's book "Data reduction and Error Analysis for the Physical Sciences", McGraw-Hill, 1969,
Arguments	See Bevington's book

The `CURFT` routine is part of the CURFIT fitting package, used for the line Gaussian fits in the gain history accumulation programs, as well as by unofficial software. It includes also the following auxiliary routines.

- ## fchi

Library	general	Fortran code
Calling sequence	See Bevington's book	
Arguments	See Bevington's book	

Chisquare computation routine in the CURFIT package.

- ## fder

Library	general	Fortran code
Calling sequence	See Bevington's book	
Arguments	See Bevington's book	

Partial derivative computation routine in the CURFIT package.

- ## mtinv

Library	general	Fortran code
Calling sequence	See Bevington's book	
Arguments	See Bevington's book	

Matrix inversion routine in the CURFIT package.

- ## dotproduct

Library	general	Fortran code
Calling sequence	`DOUBLE PRECISION val=DOTPRODUCT(a,b,n)`	
Arguments	`DOUBLE PRECISION`	`A(*),B(*)`
	`INTEGER`	`N`

A trivial function which **returns** the dot product of two arrays `A` and `B` of dimension `N`.

- ## extrp

A couple of trivial functions which return the linear interpolation at coordinate `X` between two points (`X1,Y1`) and (`X2,Y2`) with a protection to avoid divide checks when X1=X2.
Comes in two flavours for different precision.

Library	general	Fortran code
Calling sequence	`REAL y=EXTRP(y1,y2,x1,x2,x)`	
Arguments	`REAL`	`X1,Y1`
	`REAL`	`X2,Y2`
	`REAL`	`X`

- ## extrpd

Library	general	Fortran code
Calling sequence	`DOUBLE PRECISION y=EXTRP(y1,y2,x1,x2,x)`	
Arguments	`DOUBLE PRECISION`	`X1,Y1`
	`DOUBLE PRECISION`	`X2,Y2`
	`DOUBLE PRECISION`	`X`

- ## free_lu

Library	general	Fortran code
Calling sequence	`CALL FREE_LU(lu)`	
Arguments	`INTEGER`	`LU`

A trivial routine which **returns** the next free (unopened) logical unit in range 1-99, or the value -1 if no free units exist.

- ## gnomonic

Library	general	Fortran code
Calling sequence	CALL GNOMONIC(TRA,TDEC,ZRA,ZDEC,CSI,ETA)	
Arguments	DOUBLE PRECISION	TRA,TDEC
	DOUBLE PRECISION	ZRA,ZDEC
	DOUBLE PRECISION	CSI,ETA

Spherical trigonometry routine which returns the gnomonic angular coordinates CSI,ETA of a target TRA,TDEC with respect to a pointing ZRA,ZDEC according to the prescriptions of C A Murray, Vectorial Astrometry, pag. 191 ff.
All angular quantities shall be in radians.

- ## hexi4

Library	general	Fortran code
Calling sequence	CALL HEXI4(hex,i4)	
Arguments	CHARACTER*(1-8)	HEX
	INTEGER	I4

A trivial routine which converts a (zero-padded on the left) hexadecimal digit HEX (as a string of 1 to 8 characters) into a 32-bit integer value I4

- ## interpolate

Library	general	Fortran code
Calling sequence	CALL INTERPOLATE(DATUM,X,Y,N)	
Arguments	REAL	DATUM
	REAL	X(*),Y(*)
	INTEGER	N

This interpolation routine takes x=DATUM, locates the two X values X(i),X(i+1) in the X array which comprise it, and **returns in the same** DATUM the y coordinate linearly interpolated between X(i),Y(i) and X(i+1),Y(i+1) using values in the Y array, and the extrp routine.
Both arrays are of dimension N.
If x=DATUM lies outside X, an extrapolation from the first or last two points is made.

- ## lowcase

Library	general	Fortran code
Calling sequence	CALL LOWCASE(string)	
Arguments	CHARACTER*(*)	STRING

Trivial routine to convert in place a STRING to lower case.

- ## matproduct

Library	general	Fortran code
Calling sequence	CALL MATPRODUCT(A,B,C,N)	
Arguments	DOUBLE PRECISION	A(n,n),B(n,n)
	DOUBLE PRECISION	C(n,n)
	INTEGER	N

A trivial routine which **returns** the matricial product C of two square arrays A and B of dimension N*N.

- ## radecroll

Library	general	Fortran code
Calling sequence	CALL RADECROLL(EULER,RA,DEC,ROLL)	
Arguments	DOUBLE PRECISION	EULER(3,3)
	DOUBLE PRECISION	RA,DEC,ROLL

Spherical trigonometry routine which **returns** a pointing RA,DEC and ROLL angle (in radians) given an input EULER matrix.

- ## ran1

Library	general	Fortran code
Calling sequence	REAL value=RAN1(ISEED)	
Arguments	INTEGER	ISEED

Random number generator of values uniformly distributed in the range 0.0-1.0. Must be initialized calling it with a negative seed and then called repeatedly to extract random values, as in the example

```
ISEED=-123456789
dummy=RAN1(ISEED)
...
DO i=1,n
VAL(i)=RAN1(ISEED)
ENDDO
```

- ## swapi2

A family of in-place byte swapping routines for 16-bit, 32-bit and 64-bit quantities. They operate on an array `DATA` of dimension `N` exploiting an equivalence with a local array of characters.

Library	general	Fortran code
Calling sequence	CALL SWAPI2(DATA,N)	
Arguments	INTEGER*2	DATA(*)
	INTEGER	N

The above version is specific of 16-bit integers.

- ## swapi4

Library	general	Fortran code
Calling sequence	CALL SWAPI4(DATA,N)	
Arguments	any*4	DATA(*)
	INTEGER	N

The 32-bit version can be user with both INTEGER and REAL arguments.

- ## swapr8

Library	general	Fortran code
Calling sequence	CALL SWAPR8(DATA,N)	
Arguments	REAL*8	DATA(*)
	INTEGER	N

The above version is specific of 64-bit real data (DOUBLE PRECISION).

- ## time_1970

A family of routines to handle conversion of times. An Unix time `I70` is defined as the 32-bit number of seconds since 1 Jan 1970. A time array `ITIME(7)` is an array containing year,month,day,hour,minute,seconds and hundredth of seconds (the latter usually zero). An ASCII time string `TIME` is in the form `YYYY-MON-DD HH:MM:SS.FF` with three-letter codes for months.

Library	general	Fortran code
Calling sequence	CALL TIME_1970(ITIME,I70)	
Arguments	INTEGER	ITIME(7)
	INTEGER	I70

The above converts from time array to Unix time.

- ## time_array

Library	general	Fortran code
Calling sequence	CALL TIME_ARRAY(I70,ITIME)	
Arguments	INTEGER	I70
	INTEGER	ITIME(7)

The above converts from Unix time to time array

- ## time_ascii

Library	general	Fortran code
Calling sequence	`CALL TIME_ASCII(I70,TIME)`	
Arguments	`INTEGER`	`I70`
	`CHARACTER*(23)`	`TIME`

The above converts from Unix time to an ASCII time string (at least 23 characters are necessary.

- **true_length**

Library	general	Fortran code
Calling sequence	`INTEGER l=TRUE_LENGTH(string)`	
Arguments	`CHARACTER*(*)`	`STRING`

A trivial routine which returns the "true" length of `STRING`, i.e. excluding any trailing blanks.

- **udouble**

Library	general	C code
Calling sequence	`DOUBLE PRECISION var=UDOUBLE(i)`	
Arguments	`INTEGER`	`I`

This routine takes a 32-bit integer `I` which might contain an unsigned value (unsupported in Fortran) and uses C to cast it into a DOUBLE PRECISION value.

In practice this is used for the support of spacecraft times.

- **upcase**

Library	general	Fortran code
Calling sequence	`CALL UPCASE(string)`	
Arguments	`CHARACTER*(*)`	`STRING`

Trivial routine to convert in place a `STRING` to upper case.

[Previous][Next] [Up][Down]

5.3 The `xaslib` library

The `xaslib` library groups all (system-independent) routines which are not mission-specific, nor topic-specific, but typically apply to the XAS data format files.

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

- ## addhistory

Library	xaslib	Fortran code
Calling sequence	CALL ADDHISTORY(string)	
Arguments	CHARACTER*(*)	STRING

Category : XAS file header keywords
This routine writes a sequence of one or more HISTORY keywords splitting the user-supplied STRING into pieces shoter or equal to 68 characters and by repeated calls to h_add_keyword
Side effects : STRING is modified to collapse multiple blanks, and the in-memory header is entirely flushed to disk (this is considered normal since this should be the last call when processing an output file).

- ## askbin

Category : Accumulation
These two routines concerning the setup of time profile accumulations are not intended to be called directly, but by satellite specific "range setup" routines like sax_acc_range and alike.

Library	xaslib	Fortran code
Calling sequence	CALL ASKBIN(DEFBIN,ZOOM)	
Arguments	INTEGER	DEFBIN
	INTEGER	ZOOM

This routine, being passed a default minimum bin size for a time profile DEFBIN (in spacecraft OBT units), asks the user for the wished bin size (in seconds), and **returns** the corresponding ZOOM factor (as multiple of the default minimum).

- ## asktime

Library	xaslib	Fortran code
Calling sequence	CALL ASKTIME(KEY,TIMED)	
Arguments	CHARACTER*(*)	KEY ('Start'|'End')
	DOUBLE PRECISION	TIMED

This routine, being passed a default time TIMED (in spacecraft OBT units ?), asks the user for the wished time (as y m d h m s f time array), and **returns** the corresponding TIMED as elapsed seconds from a reference time.
The call must be done separately for start and end times of an accumulation using the appropriate value of KEY

- ## blkbincommon

Library	xaslib	Fortran code
Calling sequence	EXTERNAL BLKBINCOMMON	

This BLOCK DATA routine is called implicitly by the create or open calls for tabular XAS files (spectra, time profiles or photon lists) to initialize the BINCOMMON common block used to keep track of binary table characteristics.

- ## blkctxcommon

Library	xaslib	Fortran code
Calling sequence	EXTERNAL BLKCTXCOMMON	

This BLOCK DATA routine is called implicitly by buildpath to initialize a small CTXCOMMON common block containing the current instrument and context codes (the context is the wished type of data to be created, e.g. spectrum, image etc.).

- ## blkhcommon

Library	xaslib	Fortran code
Calling sequence	EXTERNAL BLKHCOMMON	

This BLOCK DATA routine is called implicitly by header read or modification routines and by the low level XAS file opening routines) to initialize the HCOMMON common block used to keep track of XAS file characteristics, inclusive of the in-memory header buffers.

- ## buildpath

Library	xaslib	Fortran code

Calling sequence	CALL BUILDPATH(NAMEIN,CODE,NAMEOUT)	
Arguments	CHARACTER*(*)	NAMEIN
	CHARACTER*(*)	CODE ('FOT'|'DATA'|'PRINT'|'CALIB')
	CHARACTER*(*)	NAMEOUT

This widely used call takes a pathless or relative filename `NAMEIN` and **returns** a full path `NAMEOUT` which locates the file in the appropriate directory according to the specification of `CODE` and also adds the appropriate file extension according to context :

- if `NAMEIN` is an absolute path, it is assumed already qualified (only file extension processing)
- for `CODE='DATA'` the path is constructed concatenating the environment variables rootdir, datadir, date, target, instrument in the order mandated by order
- for `CODE='FOT'` the path is constructed similarly but using also fotdir and fotorder
- for `CODE='PRINT'` the path is constructed similarly but using printdir and printorder
- for `CODE='CALIB'` the file is searched in the private directory pointed by the user defined environment variable mycaldir and, if such variable is not defined, or the specific file is not present, in the spacecraft and instrument specific subdirectory of the calibration directory

---

- ## close_xas_file

Library	xaslib	Fortran code
Calling sequence	CALL CLOSE_XAS_FILE(lu)	
Arguments	INTEGER	LU

This routine closes the XAS file open on logical unit `LU`, flushing its header to disk, and freeing the associated memory buffers.
A call to this routine is not necessary if the header is flushed to disk by other means and the same XAS file number will not be reused in the same program.
It is recommended to close XAS files in the reverse order in which they were opened (this is a feature of the association between logical units and XAS file numbers).

- ## copy_table_desc

Library	xaslib	Fortran code
Calling sequence	CALL COPY_TABLE_DESC (in,out)	
Arguments	INTEGER	IN,OUT

Copies the BINCOMMON table descriptor for table number `IN` into the one for table number `OUT`, where `IN,OUT` are XAS file numbers.

- ## create_image

Library	xaslib	Fortran code
Calling sequence	CALL CREATE_IMAGE(lu,filename,sizex,sizey,array,nx,ny,type)	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	SIZEX,SIZEY
	any*4	ARRAY
	INTEGER	NX,NY
	CHARACTER*(3)	TYPE ('FLO'|'INT')

This routine creates and writes a new XAS image file with VOS name `FILENAME` using logical unit `LU`. It writes an image of logical sizes `SIZEX,SIZEY` taking it from the array `ARRAY` of physical sizes `NX,NY` (the distinction hardly matters if one uses dynamic memory allocation).
`ARRAY` is normally REAL (`TYPE='FLO'`), although the routine will correctly write it even if INTEGER*4, however the `TYPE='INT'` is not honored at present (the BITPIX keyword is set to -32, i.e. real data).
An image file written this way shall be immediately usable (has a minimal header), however it is recommended to complete writing of header keywords.

- ## create_photon

Library	xaslib	Fortran code
Calling sequence	CALL CREATE_PHOTON(lu,filename,nbins,izoff)	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	NBINS
	INTEGER	IZOFF

This routine creates a new XAS photon file with VOS name `FILENAME` using logical unit `LU`. It reserves space for `NBINS` records (photons) and **returns** the zero-record offset `IZOFF` to be passed to other calls like write_bin.
It prepares a minimal header with the binary table keywords according to the content of a table

descriptor prepared in advance, but does not write any data.

- ## create_spectrum

Library	xaslib	Fortran code
Calling sequence	`CALL CREATE_SPECTRUM(lu,filename,nbins,izoff)`	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	NBINS
	INTEGER	IZOFF

This routine creates a new XAS spectrum file with VOS name `FILENAME` using logical unit `LU`. It reserves space for `NBINS` records (channels) and **returns** the zero-record offset `IZOFF` to be passed to other calls like write_bin or used as offset to write records directly.
It prepares a minimal header with the binary table keywords according to the content of a table descriptor prepared in advance, but does not write any data.

- ## create_time

Library	xaslib	Fortran code
Calling sequence	`CALL CREATE_TIME(lu,filename,nbins,izoff)`	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	NBINS
	INTEGER	IZOFF

This routine creates a new XAS time profile with VOS name `FILENAME` using logical unit `LU`. It reserves space for `NBINS` records (time bins) and **returns** the zero-record offset `IZOFF` to be passed to other calls like write_bin.
It prepares a minimal header with the binary table keywords according to the content of a table descriptor prepared in advance, but does not write any data.

- ## decodetform

Library	xaslib	Fortran code
Calling sequence	`CALL DECODETFORM(TFORM,IBIT,N)`	
Arguments	CHARACTER*(*)	TFORM
	INTEGER	IBIT
	INTEGER	N

Category: binary table support
This routines receives a FITS-style TFORM keyword (only 'nB','nI','nJ','nE','nD' are recognised) and **returns** the corresponding number of bits `IBIT` (in FITS BITPIX usage, i.e. 8,16,32,-32,-64) and depth (`N=n`)

- ## depath

Library	xaslib	Fortran code
Calling sequence	`CALL DEPATH(NAME)`	
Arguments	CHARACTER*(*)	NAME

This routine **modifies in place** the filename `NAME` stripping its path and the file extension.

- ## depath_1

Library	xaslib	Fortran code
Calling sequence	`CALL DEPATH_1(NAME)`	
Arguments	CHARACTER*(*)	NAME

This routine **modifies in place** the filename `NAME` stripping its path but leavin the file extension intact.

- ## detpointing

Library	xaslib	Fortran code
Calling sequence	`CALL DETPOINTING(RA,DEC,ROLL)`	
Arguments	DOUBLE PRECISION	RA,DEC,ROLL

Category: attitude
Side effects: loading info in PIXCOMMON
This routine **returns** in radians the detector pointing coordinates RA, DEC and ROLL angle. If they are present in the current image file header, it just returns them.
Otherwise it calls misalign to get the misalignments, satpointing to get the spacecraft pointing, computes Euler angles and applies the rotation matrix with radecroll

---

- ## get_global_default

Library	xaslib	Fortran code
Calling sequence	CALL GET_GLOBAL_DEFAULT(name,default,value)	
Arguments	CHARACTER*(*)	NAME
	CHARACTER*(*)	DEFAULT
	CHARACTER*(*)	VALUE

Category: environment access
This is the standard recommended way to obtain the VALUE of an enviroment variable of given NAME. It calls the VOS z_get_global routine but, if the variable does not exist, returns the user supplied DEFAULT

- ## get_obs_chain

Library	xaslib	Fortran code
Calling sequence	CALL GET_OBS_CHAIN(IOBS,KEEP)	
Arguments	INTEGER	IOBS
	CHARACTER*(5)	KEEP ('Keep'|'Reset'|'Any')

Category:accumulation support
This routine is called by (SAX packetcap based) telemetry reading routines to know what is the next observation in the current chain (stored in the environment). The observation number is returned in IOBS, or zero if there is no further observation.

- KEEP='Reset' returns in IOBS the first observation of the chain.
- KEEP='Keept' returns in IOBS the current observation of the chain and does not advance the observation pointer (the same IOBS will be returned next time)
- any other value advances the observation pointer

- ## get_table_desc

Library	xaslib	Fortran code
Calling sequence	CALL GET_TABLE_DESC (table,column,offset,bitpix,dimens)	
Arguments	INTEGER	TABLE
	INTEGER	COLUMN
	INTEGER	OFFSET,BITPIX,DIMENS

Category: binary table support
This routines **returns** the characteristics (i.e. the physical column number OFFSET, the bit width in FITS syntaxBITPIX and the depth or dimensionality DIMENS) of the column at logical number COLUMN in the binary table file with XAS file number tt>TABLE, reading it from the BINCOMMON common block.

- ## getmisalign

Library	xaslib	Fortran code
Calling sequence	CALL GETMISALIGN(ALPHA,BETA,GAMMA)	
Arguments	DOUBLE PRECISION	ALPHA,BETA,GAMMA

Category: attitude
This routine returns the Z,Y and X misalignments ALPHA,BETA,GAMMA reading them from the appropriate calibration file or asking the user.
Side effect: loads instrument focal length in COMMON block.

- ## h_add_keyword

This routine has 5 entry points and is used to add an header keyword of appropriate type, given NAME and VALUE. For all calls one can specify if the keyword is just added to the in-memory copy of the header (FLUSHFLAG=0) or if the header is also flushed to disk (FLUSHFLAG=0). It is suggested to defer flushing only to significant points (adding a complete series of keywords, or closing the file).

Library	xaslib	Fortran code

	CHARACTER*(*)	VALUE
	INTEGER	FLUSHFLAG (0|1)

The above call is for CHARACTER keywords (i.e. VALUE is a string)

Calling sequence	CALL H_ADD_IKEYWORD(name,value,flushflag,n)	
Arguments	INTEGER*2	VALUE(*)
	INTEGER	N

The above call is for 16-bit integer keywords (discouraged). See below for the explanation of the extra arguments and above for the common arguments (not shown).

Calling sequence	CALL H_ADD_JKEYWORD(name,value,flushflag,n)	
Arguments	INTEGER	VALUE(*)
	INTEGER	N

The above call is for 32-bit integer keywords. In this case, as for all numeric keywords, the values can be an array VALUE(*) of at least N elements. If VALUE is a scalar, N shall be a variable with value 1. N shall always be a variable.

Calling sequence	CALL H_ADD_RKEYWORD(name,value,flushflag,n)	
Arguments	REAL	VALUE(*)
	INTEGER	N

Similar call for numeric 32-bit floating point REALs

Calling sequence	CALL H_ADD_DKEYWORD(name,value,flushflag,n)	
Arguments	DOUBLE PRECISION	VALUE(*)
	INTEGER	N

Similar call for numeric 64-bit floating point DOUBLE PRECISION values.

- ## h_copy_all_header

Library	xaslib	Fortran code
Calling sequence	CALL H_COPY_ALL_HEADER(in,out)	
Arguments	INTEGER	IN,OUT

Copies the in-memory header for XAS file number IN into the one for XAS file number OUT. The headers are not flushed to disk. The destination is overwritten entirely (thus any keyword which shall be different must be saved and restored care of the program).

- ## h_current_file

Library	xaslib	Fortran code
Calling sequence	CALL H_CURRENT_FILE(XASNO)	
Arguments	INTEGER	XASNO

Most header operations (typically all keyword access routines) operate on the current XAS file (typically the last one opened). This routine allows to switch future operations to another file identified by its XAS file number XASNO.

- ## h_find_keyword

Library	xaslib	Fortran code
Calling sequence	CALL H_FIND_KEYWORD(name,found,type,length,nelem,pointer)	
Arguments	CHARACTER*(*)	NAME
	LOGICAL	FOUND
	INTEGER	TYPE
	INTEGER	LENGTH
	INTEGER	NELEM
	INTEGER	POINTER

This dedicated routine is not intended for general use. It locates a kewyord with given NAME in the memory copy of the header. A successful search returns FOUND=.TRUE.. In such case TYPE is the keyword type (0 chracter, 1 INTEGER*2, 2 INTEGER*2, 3 REAL*4, 4 REAL*8) LENGTH is the number of bytes occupied by the data field value in the memory buffer, NELEM is the number of elements (in arrays, 1 is scalar) and POINTER is the location of the keyword in the memory buffer
The scope of the search can be controlled by the HCOMMON_CONTEXT variable (since this is not intended for public use, it is documented only in the code)

• h_flush_header

Library	xaslib	Fortran code
Calling sequence	`CALL H_FLUSH_HEADER`	

This service routine flushes the in-memory header buffer to disk. Not intended for public use.

• h_flush_minih

Library	xaslib	Fortran code
Calling sequence	`CALL H_FLUSH_MINIH(zoff)`	
Arguments	INTEGER	ZOFF

This service routine flushes the file mini-header buffer to disk. Not intended for public use. Called by the previous routine.
The mini-header is represented by 28 bytes of information which are always stored at the beginning of the file (the remainder of the header is stored after the data, being actually a trailer), and spans `ZOFF` records (typically 1, but more if the file record length is very short).

• h_load_header

Library	xaslib	Fortran code
Calling sequence	`CALL H_LOAD_HEADER`	

This service routine reads the in-memory header buffer from disk. Not intended for public use. It takes also care of initial dynamic memory allocation for the header.

• h_load_minih

Library	xaslib	Fortran code
Calling sequence	`CALL H_LOAD_MINIH(recl,zoff)`	
Arguments	INTEGER	RECL
	INTEGER	ZOFF

This service routine reads a file mini-header from disk. Not intended for public use.
The mini-header is described above and spans `ZOFF` records. This call is usually the first after a file has been opened (opening makes known the file record length in bytes `RECL`, required by this routine).

• h_modify_keyword

This routine, similary to h_add_keyword, has 5 entry points and is used to modify an header keyword of appropriate type, given `NAME` and `VALUE`. The calling sequence is the same used for h_add_keyword, to which one is referred for all details.
In fact this routine calls h_add_keyword if a keyword with given `NAME` does not exist. Otherwise it locates the existing keyword and changes its value (but cannot extend the data field length, therefore a string keyword cannot be made longer, and an array keyword cannot have more elements)
Normal keywords are not "duplicatable". Only a given keyword with a given `NAME` can be present in a file. There are however some "duplicatable" keywords (namely COMMENT, HISTORY and PARENTS which cannot be modified by this routine (a new keyword will always be added)

Library	xaslib	Fortran code
Calling sequence	`CALL H_MODIFY_KEYWORD(name,value,flushflag)`	
	`CALL H_MODIFY_IKEYWORD(name,value,flushflag,n)`	
	`CALL H_MODIFY_JKEYWORD(name,value,flushflag,n)`	
	`CALL H_MODIFY_RKEYWORD(name,value,flushflag,n)`	
	`CALL H_MODIFY_DKEYWORD(name,value,flushflag,n)`	

For character, INTEGER*2, INTEGER*4, REAL and DOUBLE PRECISION keywords respectively

---

• h_next_keyword

Library	xaslib	Fortran code
Calling sequence	`CALL H_NEXT_KEYWORD(NAME,NORDER,POINTER)`	
Arguments	CHARACTER*(*)	NAME
	INTEGER	NORDER
	INTEGER	POINTER

This dedicated routine is not intended for general use and is used only by some specific programs which need to scan the header from the beginning. It **returns** the `NAME` of the next keyword (or a null, i.e. `CHAR(0)` if there are no more keywords), its sequence number `NORDER` and the location of the keyword in the memory buffer POINTER.

- ## h_read_keyword

This routine has 5 entry points and is used to retrieve the VALUE of an header keyword of appropriate type, given its NAME. For all calls, if the named keyword is not found, an user supplied DEFAULT is returned in VALUE.

All calls return also an ERROR code, which is 0 for no errors, 1 for keyword not found, 2 for type mismatch (keyword is not of the type implied by the call), 3 if the header has not yet been loaded from disk, and -1 for a truncation error (the space provided in VALUE is not enough to contain the actual value). Error codes less or equal to zero imply that the partial or full value is returned. Error codes greater than zero imply that DEFAULT is returned in its stead.

Library	xaslib	Fortran code
Calling sequence	CALL H_READ_KEYWORD(name,value,default,error)	
Arguments	CHARACTER*(*)	NAME
	CHARACTER*(*)	VALUE
	CHARACTER*(*)	DEFAULT
	INTEGER	ERROR

The above call is for CHARACTER keywords (i.e. VALUE is a string)

Calling sequence	CALL H_READ_IKEYWORD(name,value,default,error,n)	
Arguments	INTEGER*2	VALUE(*)
	INTEGER*2	DEFAULT
	INTEGER	N

The above call is for 16-bit integer keywords (discouraged). See below for the explanation of the extra arguments and above for the common arguments (not shown).

Calling sequence	CALL H_READ_JKEYWORD(name,value,default,error,n)	
Arguments	INTEGER	VALUE(*)
	INTEGER	DEFAULT
	INTEGER	N

The above call is for 32-bit integer keywords. In this case, as for all numeric keywords, the values can be an array VALUE(*) of at least N elements. N shall always be a variable initialized to the number of wished elements to be retrieved. If VALUE is a scalar, N shall be a variable with value 1. This because N is **modified** by the program which **returns** the actual number of keywords retrieved : if the array is longer, no more than than the requested N are returned, but if it is shorter only the first actual N are returned. The rest of the VALUE(*) array is filled with the (scalar) DEFAULT (which in case of errors is used to fill the entire array).

Calling sequence	CALL H_READ_RKEYWORD(name,value,default,error,n)	
Arguments	REAL	VALUE(*)
	REAL	DEFAULT
	INTEGER	N

Similar call for numeric 32-bit floating point REALs

Calling sequence	CALL H_READ_DKEYWORD(name,value,default,error,n)	
Arguments	DOUBLE PRECISIO	VALUE(*)
	DOUBLE PRECISIO	DEFAULT
	INTEGER	N

Similar call for numeric 64-bit floating point DOUBLE PRECISION values.

- ## h_update_datasize

Library	xaslib	Fortran code
Calling sequence	CALL H_UPDATE_DATASIZE(NDATA)	
Arguments	INTEGER	NDATA

This routine shall be used in the following case :
  - one has created a XAS data file reserving space for a number of records (or even for zero records)
  - this has implicitly written to disk the header after the data area for such records (which can also be filled later)
  - one has either written more records, or decided that less records have to be used (in which case the disk header is at the wrong place) where NDATA is the final number of actual records
  - one might have also updated the in-memory copy of the header

In all this cases one must call this routine to update the 28-byte mini-header, change the NAXIS2 keyword (which for all XAS files is the number of records), and flush the header to disk at the correct place.

- **init_timewindow**

Library	xaslib	Fortran code
Calling sequence	CALL INIT_TIMEWINDOW(FILE)	
Arguments	CHARACTER*(*)	FILE

Category: time window / accumulation support
The routine retrieves the name of the timewindow file from the environment and **returns** it in FILE (a blank string is returned if no timewindow is defined).
If a file is defined it calls openwindow and loadwindow to read the file (an ASCII file containing a list of start and end times in time array form) and store the times in a COMMON block in the appropriate time units.

- **leftnumber**

Library	xaslib	Fortran code
Calling sequence	CHARACTER*3 FUNCTION LEFTNUMBER(N,ISIZE)	
Arguments	INTEGER	N
	INTEGER	ISIZE

Category: binary table support
This function formats a number N (0-999) into a 3-digit left-justified, blank padded string (i.e. '1 ','10 ','100') and returns in ISIZE the number of non-blank digits (in the example respectively 1,2,3), or zero in case of errors (including N out of range) when the string '***' is returned.
This routine is used by maketform to build FITS-style TFORM values.

- **loadwindow**

Library	xaslib	Fortran code
Calling sequence	CALL LOADWINDOW(LU,N,START,END)	
Arguments	INTEGER	LU
	INTEGER	N
	DOUBLE PRECISION	START(*),END(*)

Category: time window
This routine reads from the timewindow file opened by openwindow on logical unit LU a number N of start and end times into the START and END arrays (as a number of seconds since 1970 but in double precision format).

- **maketform**

Library	xaslib	Fortran code
Calling sequence	CHARACTER*(*) FUNCTION MAKETFORM(IBIT,N)	
Arguments	INTEGER	IBIT
	INTEGER	N

Category: binary table support
This routine receives the number of bits IBIT (in FITS BITPIX usage, i.e. 8,16,32,-32,-64) and depth (N=n) of a binary table column and returns a formatted FITS-style TFORM keyword (only 'nB','nI','nJ','nE','nD' are supported in XAS)

- **misalign**

Library	xaslib	Fortran code
Calling sequence	CALL MISALIGN(EULER,NEWMAT)	
Arguments	DOUBLE PRECISION	EULER(3,3)
	DOUBLE PRECISION	NEWMAT(3,3)

Category: attitude
This routines takes an input attitude EULER matrix (typically the spacecraft attitude) and rotates it by the Z,Y,X misalignments (retrieved via getmisalign producing a new attitude matrix tt>NEWMAT

- **multiply_rmfarf**

Library	xaslib	Fortran code

	REAL	ARF(NEBN)
	REAL	ELOW(NEBN),EUP(NEBN)
	INTEGER	NEBN,NEOUT

This routine takes a "pure" (adimensional) response matrix function from RMF, and returns in the same RMF the product (cm^2 keV) of it by the ARF (cm^2) and by the energy grid bin width (computed from the lower and upper energy bounds in ELOW and EUP. The units are consistent with the XAS convention for response matrices.
NEBN is the number of input energies, while NEOUT is the number of PHA channels.

- ## open_image

Library	xaslib	Fortran code
Calling sequence	CALL OPEN_IMAGE(lu,filename,sizex,sizey,bitpix,izoff)	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	BITPIX
	INTEGER	SIZEX,SIZEY
	INTEGER	IZOFF

This routine opens an existing XAS image file with VOS name FILENAME using logical unit LU. It **returns** the image size SIZEX,SIZEY, the number of bits/pixel BITPIX (in FITS convention, -32 for floating point images is the recommended usage) and the zero-record offset IZOFF to be passed to other calls like read_image.

- ## open_matrix

Library	xaslib	Fortran code
Calling sequence	CALL OPEN_MATRIX(lu1,lu2,filename,sizex,sizey,izoff1,izoff2)	
Arguments	INTEGER	LU1,LU2
	CHARACTER*(*)	FILENAME
	INTEGER	SIZEX,SIZEY
	INTEGER	IZOFF1,IZOFF2

This routine opens simulatenously on logical units LU1,LU2 a response matrix (a floating point XAS image file) with VOS name FILENAME and its associated histogram (the associated histogram is a 1-d XAS image file, containing the input energy grid, and whose name is stored in a keyword in the matrix header). It **returns** the matrix size SIZEX,SIZEY and the zero-record offsets IZOFF1,IZOFF2 to be passed to other calls like read_image to actually read the matrix and histogram.

- ## open_new_xas_file

Library	xaslib	Fortran code
Calling sequence	CALL OPEN_NEW_XAS_FILE(lu,file,type,recl,nrec,xasno,zoff)	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILE
	CHARACTER*(*)	TYPE
	INTEGER	RECL
	INTEGER	NREC
	INTEGER	XASNO
	INTEGER	ZOFF

This routine (normally not called by users, which use the higher level type-specific "create" routines like create_image) creates (associating it with logical unit LU) a new XAS file of name FILE and given TYPE with NREC data records of record length of RECL bytes (the number of actual records is larger, since it includes the header and mini-header). It **returns** the XAS file number XASNO and the zero-offset record ZOFF after which data can be written.
Recognised file types are :
- 'FLO' for normal real images,
- 'INT' for normal integer images (deperecated)
- 'MAT' for response matrix images
- 'SPE' for spectra (binary tables)
- 'TIM' for time profiles (binary tables)
- 'PHO' for photon files (binary tables)
- 'GEN' for generic binary tables

- ## open_old_xas_file

Library	xaslib	Fortran code

Calling sequence	`CALL OPEN_OLD_XAS_FILE(lu,file,type,recl,nrec,xasno,zoff)`	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILE
	CHARACTER*(*)	TYPE
	INTEGER	RECL
	INTEGER	NREC
	INTEGER	XASNO
	INTEGER	ZOFF

This routine (normally not called by users, which use the higher level type-specific "open" routines like open_image or open_photon) opens (associating it with logical unit `LU`) an existing XAS file of name `FILE`, **returning** its `TYPE`, the number of data records `NREC` records, and the record length of `RECL` bytes. It **returns** also the XAS file number `XASNO` and the zero-offset record `ZOFF` after which data can be written. Note that the syntax (except for the argument intent) is identical to the one of open_new_xas_file to which one is referred (except that any non-matrix image is returned as `TYPE='IMG'`).

- ## open_photon

Library	xaslib	Fortran code
Calling sequence	`CALL OPEN_PHOTON(lu,filename,recl,nbins,nfields,izoff)`	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	RECL
	INTEGER	NBINS
	INTEGER	NFIELDS
	INTEGER	IZOFF

This routine opens an existing XAS photon list file with VOS name `FILENAME` using logical unit `LU`. It **returns** the number of photons `NBINS`, the number of data fields (columns) per event `NFIELDS`, the record length `RECL` in bytes, and the zero-record offset `IZOFF` to be passed to other calls like read_bin. It also loads a table descriptor according to the characteristics of the table columns (fields), in the order in which they appear, ignoring any unnamed physical column (TTYPE blank) which is used for pad columns.

- ## open_spectrum

Library	xaslib	Fortran code
Calling sequence	`CALL OPEN_SPECTRUM(lu,filename,recl,nbins,nfields,izoff)`	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	RECL
	INTEGER	NBINS
	INTEGER	NFIELDS
	INTEGER	IZOFF

This routine opens an existing XAS spectrum file with VOS name `FILENAME` using logical unit `LU`. It **returns** the number of channels `NBINS`, the number of data fields (columns) `NFIELDS` (usually 4), the record length `RECL` in bytes, and the zero-record offset `IZOFF` to be passed to other calls like read_bin or used to read the records directly. It also loads a table descriptor according to the characteristics of the table columns (fields), only for the 4 columns with canonic names in a spectrum.

- ## open_time

Library	xaslib	Fortran code
Calling sequence	`CALL OPEN_TIME(lu,filename,recl,nbins,nfields,izoff)`	
Arguments	INTEGER	LU
	CHARACTER*(*)	FILENAME
	INTEGER	RECL
	INTEGER	NBINS
	INTEGER	NFIELDS
	INTEGER	IZOFF

This routine opens an existing XAS time profile (light curve file) with VOS name `FILENAME` using logical unit `LU`. It **returns** the number of time bins `NBINS`, the number of data fields (columns) per bin `NFIELDS`, the record length `RECL` in bytes, and the zero-record offset `IZOFF` to be passed to other calls like read_bin. It also loads a table descriptor according to the characteristics of the table columns (fields), assigning logical column numbers only for the those columns actually present whose name is one of the canonic names in a light curve (or in unofficially supported folded light curves).

- **open_xas_ascii**

Library	xaslib	Fortran code
Calling sequence	CALL OPEN_XAS_ASCII(LU,NAME,NROW,NCOL)	
Arguments	INTEGER	LU
	CHARACTER*(*)	NAME
	INTEGER	NROW
	INTEGER	NCOL

This routine opens an existing "XAS ASCII" tabular file of given NAME on logical unit LU. Such file is just a plain ASCII file with NROW records with NCOL numeric columns, preceded by as many as wished nh header records in free format, preceded by a single pseudo-miniheader line in the form XAS1ASC2GEN31234 *nh nrow ncol*. The routine skips the nh+1 header records and stays positioned for reading the first data record, at the same time **returning** NROW and NCOL.

There is instead no explicit routine interface to create new "XAS ASCII" tabular files.

- **openwindow**

Library	xaslib	Fortran code
Calling sequence	CALL OPENWINDOW(LU,NAME,N)	
Arguments	INTEGER	LU
	CHARACTER*(*)	NAME
	INTEGER	N

Category: time window
This routine is used to open a timewindow file with given NAME on logical unit LU before calling loadwindow. The routine does a consistency format check on the content of each time window. It **returns** the number of timewindows N or N=0 in case of any error.

- **pad_table**

Library	xaslib	Fortran code
Calling sequence	CALL PAD_TABLE(TABLE,PADCOL,PADDED)	
Arguments	INTEGER	TABLE
	INTEGER	PADCOL
	LOGICAL	PADDED

This service routine shall be called during creation of new XAS binary table files, to ensure that the record length is padded to a multiple of 4 bytes (constraint imposed for Fortran direct access support on Digital systems). One passes the XAS file number TABLE, and the number of a non-existing free column PADCOL (i.e. the first unused column). The routine reads the table descriptor, adds the column sizes to verify the toal size is a multiple of 4 bytes, and if not adds to the descriptor a pad column of type 1B, 2B or 3B as appropriate. It returns a logical flag PADDED=.TRUE. if the pad column has been added.

- **pipeexist**

Library	xaslib	Fortran code
Calling sequence	IF(PIPEEXIST(pipe)) THEN ..	
Arguments	CHARACTER*(*)	PIPE

This logical function is a convenience utility front end to the VOS z_channel routine to test the existence of a named communication channel PIPE.

- **pktcap_load**

Library	xaslib	Fortran code
Calling sequence	CALL PKTCAP_LOAD(SATELLITE,INSTRUMENT,MODE,PACKET)	
Arguments	CHARACTER*(*)	SATELLITE
	CHARACTER*(*)	INSTRUMENT
	CHARACTER*(*)	MODE
	CHARACTER*(*)	PACKET

Category: accumulation support
This routine is a mission-independent way to load mission-specific information about a named telemetry PACKET. It opens the appropriate packetcap file, whose name is built according to SATELLITE, INSTRUMENT and MODE (typically one of 'DIR'|'INDIR'|'HK'), and which is located in the appropriate calibration directory.

Side effect:loads in COMMON block PKCOMMON the entire reconstructed content of the packetcap entry for the named PACKET, inclusive of the resolution of any tc reference (similar tp termcap tc capabilities).

- ## pktcap_lookup

Library	xaslib	Fortran code
Calling sequence	CALL PKTCAP_LOOKUP(FIELD,TYPE,IVAL,STRVAL,FOUND)	
Arguments	CHARACTER*(*)	FIELD
	INTEGER	TYPE
	INTEGER	IVAL
	CHARACTER*(*)	STRVAL
	LOGICAL	FOUND

Category: accumulation support
This routine scans the packetcap entry currently loaded in memory by the last call to pktcap_load for a named FIELD. It **returns** a TYPE which can be :
  - 0 for a boolean field, in which case FOUND=.TRUE. means the field is set.
  - 1 for a numeric field, in which case the field value is in IVAL
  - 2 for a string field, in which case the field value is in STRVAL
  - -1 in case of errors

- ## preparse

Library	xaslib	Fortran code
Calling sequence	CHARACTER*(*+2) OUT=PREPARSE(STRING)	
Arguments	CHARACTER*(*)	STRING

Category: user interface
This function is widely used after a call to x_read which returns an argument STRING containing one or more blank or comma separated character items. It parses a string of the form AAA,BBB,CCC or AAA BBB CCC **returning** the corresponding string 'AAA','BBB','CCC' / which is suitable for list-directed reading.
The receiving variable OUT shall have enough space to contain the expanded value. It is customary to allocate at least two characters more (at least for the blank-slash list-directed terminator added at the end).

- ## read_bin

Library	xaslib	Fortran code
Calling sequence	CALL READ_BIN(LU,IREC,IZOFF,F1,F2,F3,F4,F5,F6,F7)	
Arguments	INTEGER	LU
	INTEGER	IREC
	INTEGER	IZOFF
	any*4	F1(*),F2(*),F3(*),F4(*),F5(*),F6(*),F7(*)

Category: binary tables
This routine reads a generic data record at position IREC (after the zero-offset IZOFF passed by the file opening routine) from the binary table open on logical unit LU. The Fi arguments can be scalar or arrays of any numeric data type, as appropriate according to the type and depth of the corresponding column in the table. All seven variables shall be supplied (seven is the maximum number of columns in a table, if more are required the code shall be recompiled !) but those unused may point to a dummy variable. Only logical columns marked as present in the table descriptor will be returned a value.
Note that byte and 16-bit columns are always returned in 32-bit INTEGER Fi.

- ## read_image

Library	xaslib	Fortran code
Calling sequence	CALL READ_IMAGE(lu,array,sizex,sizey,nx,ny,izoff)	
Arguments	INTEGER	LU
	REAL	ARRAY
	INTEGER	SIZEX,SIZEY
	INTEGER	NX,NY
	INTEGER	IZOFF

This routine reads an entire XAS image file opened on logical unit LU by open_image which has also passed the zero-offset record IZOFF. It fills an image of logical sizes SIZEX,SIZEY taking it from the array ARRAY of physical sizes NX,NY (the distinction hardly matters if one uses dynamic memory allocation).
The routine supports only REAL*4 floating point images (other deprecated types can be dealt with by the user directly in Fortran).

- ## satpointing

Library	xaslib	Fortran code
Calling sequence	`CALL SATPOINTING(RA,DEC,ROLL)`	
Arguments	`DOUBLE PRECISION`	`RA,DEC,ROLL`

Category: attitude
This routine **returns** in radians the satellite pointing coordinates `RA,DEC` and `ROLL` angle. If they are present in the current image file header, it just returns them, otherwise asks the user.

- ## set_table_desc

Library	xaslib	Fortran code
Calling sequence	`CALL SET_TABLE_DESC(table,column,offset,bitpix,dimens)`	
Arguments	`INTEGER`	`TABLE`
	`INTEGER`	`COLUMN`
	`INTEGER`	`OFFSET,BITPIX,DIMENS`

Category: binary table support
This routines saves the characteristics (i.e. the physical column number `OFFSET`, the bit width in FITS syntax`BITPIX` and the depth or dimensionality `DIMENS`) of the column at logical number `COLUMN` of the binary table file with XAS file number tt>TABLE into a descriptor in the BINCOMMON common block.

- ## skytoxy

Library	xaslib	Fortran code
Calling sequence	`CALL SKYTOXY(TRA,TDEC,ZRA,ZDEC,ZROLL,X,Y)`	
Arguments	`DOUBLE PRECISION`	`TRA,TDEC`
	`DOUBLE PRECISION`	`ZRA,ZDEC,ZROLL`
	`DOUBLE PRECISION`	`X,Y`

Category: attitude
This routine receives (in radians) the celestial coordinates of an object `TRA,TDEC` and a (detector) attitude `ZRA,ZDEC,ZROLL` and **returns** the `X,Y` coordinates in mm on the focal plane (pixelization is left to the user). It firsts converts in gnomonic angular coordinates, then rotates them by the roll angle, and then use the platescale to convert into mm.

- ## time_constants_setup

Library	xaslib	Fortran code
Calling sequence	`CALL TIME_CONSTANTS_SETUP(SCLSB)`	
Arguments	`INTEGER`	`SCLSB`

Category: accumulation support
This routine prepares all quantity necessary for conversion between on board times and meaningful time units (default is seconds, but can be controlled by the timeunits environment variable). Two kind of on board times are dealt with, those in current packet units (whose time resolution is derived internally via a packetcap lookup of the `tr` field), and those in spacecraft units (whose time resolution is derived by the `SCLSB` argument (negative, e.g. -16) as 2^{SCLSB} s) Side effects: stores relevant values in TIMECOMMON

- ## trimroot

Library	xaslib	Fortran code

Unsupported (variant of depath ?)

- ## update_start_end

Library	xaslib	Fortran code
Calling sequence	`CALL UPDATE_START_END`	

Category: keyword header support
This routine is called by accumulation program just before writing the HISTORY keyword, to write in the HISTORY the correct start and end times, whose values in COMMON may have been altered during the processing.

- ## voserror

Library	xaslib	Fortran code

Calling sequence	`IF( VOSERROR (ivoserr,isyserr) ) THEN ...`	
Arguments	INTEGER	IVOSERR
	INTEGER	ISYSERR

This logical function shall be called after each routine which sets a standard error code in VOSCOMMON to branch to an error message or handler in case of error (returns `.TRUE.` if error occurred). It also **returns** (extracting it from the common block) the standard (VOS) error code `IVOSERR` and the corresponding system-dependent code `ISYSERR`.
VOS error codes can be looked at in the error code listings.

- ## write_bin

Library	xaslib	Fortran code
Calling sequence	`CALL WRITE_BIN(LU,IREC,IZOFF,F1,F2,F3,F4,F5,F6,F7)`	
Arguments	INTEGER	LU
	INTEGER	IREC
	INTEGER	IZOFF
	any*4	F1(*),F2(*),F3(*),F4(*),F5(*),F6(*),F7(*)

Category: binary tables
This routine writes a generic data record at position `IREC` (after the zero-offset `IZOFF` passed by the file opening routine) into the binary table open on logical unit `LU`. The `Fi` arguments can be scalar or arrays as explained for read_bin. Only logical columns marked as present in the table descriptor will be written. Note that byte and 16-bit columns are converted care of this routine from 32-bit INTEGER `Fi`.

- ## x_prompt

Library	xaslib	Fortran code
Calling sequence	`CALL X_PROMPT(string,k)`	
Arguments	CHARACTER*(*)	STRING
	INTEGER	K

Category: user interface
This routine is a replacement for the `WRITE(*,*)'prompt'` idiom to write a terminal prompt. Note that prompts are always echoed to the terminal even if the input is taken from command file or run string arguments, unless the environment variable echo disables it.
The prompt is passed in `STRING` and issued without advancing to the next line so that the reply will be on the same line, starting at column `K` (or immediately after the prompt if `K=0`). The idiom is :

```
WRITE(BUFFER,*)' Enter your value [df=',I,'] '
CALL X_PROMPT(BUFFER,0)
```

- ## x_read

Library	xaslib	Fortran code
Calling sequence	`CALL X_READ(npar,mpar,string)`	
Arguments	INTEGER	NPAR,MPAR
	CHARACTER*(*)	STRING

Category: user interface
This routine is a replacement for the `READ(*,*)values` idiom to read an user reply to a terminal prompt. The routine can read from the terminal, from a command file, or from the run string positional parameters as controlled by environment variables TBD REF). The user can ask for `MPAR` parameters (usually 1) asking at a specific position `NPAR` (or just the next parameter if `NPAR=0`) and the routine returns the parameters in `STRING`.
`STRING` is suitable for list-directed i/o for numeric values, but must be preparsed for list-directed i/o for string values, specially for multiple strings, which are also allowed, according to the following idioms :

```
CALL X_PROMPT('Enter one integer and two reals ',0)
CALL X_READ(0,3,BUFFER)
READ(BUFFER,*)I,R1,R2

CALL X_PROMPT('Enter filename ',0)
CALL X_READ(0,1,BUFFER)
BUFFER2=PREPARSE(BUFFER)
READ(BUFFER2,*)NAME

CALL X_PROMPT('Enter two strings ',0)
CALL X_READ(0,2,BUFFER)
BUFFER2=PREPARSE(BUFFER)
READ(BUFFER2,*)STRING1,STRING2
```

- ## xasmatout

Library	xaslib	Fortran code
Calling sequence	`CALL XASMATOUT(LU1,LU2,NAME,RMF,ELOW,EUP,NEBN,NEOUT)`	
Arguments	INTEGER	LU1,LU2
	CHARACTER*(*)	NAME
	REAL	RMF(NEBN,NEOUT)
	REAL	ELOW(NEBN),EUP(NEBN)
	INTEGER	NEBN,NEOUT

This utility routine writes a XAS response matrix `RMF` into a `.mat` file named `NAME`. It uses two logical units `LU1,LU2`, where the second is for the associated histogram file which contains the middle points of the energy grid (i.e. the mean of the low and upper bounds `ELOW` and `EUPP`). Both files are written via create_image
`NEBN` is the number of input energies, while `NEOUT` is the number of PHA channels.

- **xytosky**

Library	xaslib	Fortran code
Calling sequence	`CALL XYTOSKY(X,Y,ZRA,ZDEC,ZROLL,TRA,TDEC)`	
Arguments	DOUBLE PRECISION	X,Y
	DOUBLE PRECISION	ZRA,ZDEC,ZROLL
	DOUBLE PRECISION	TRA,TDEC

Category: attitude
This routine receives the `X,Y` coordinates in mm of point on the focal plane (de-pixelization is left to the user) and a (detector) attitude `ZRA,ZDEC,ZROLL` (in radians) and **returns** the celestial coordinates of an object `TRA,TDEC` (also in radians).

- **zx_get_parameter**

Library	xaslib	Fortran code
Calling sequence	`CALL ZX_GET_PARAMETER(npar,string,length)`	
Arguments	INTEGER	NPAR
	CHARACTER*(*)	STRING
	INTEGER	LENGTH

Category: user interface
This routine is not normally called by users, which use x_read (which calls this) instead (unless they need specific repeated access to a positional parameter on the runstring and are really sure of what they do). The routine retrieves (from runstring, command file or terminal) a single parameter at position `NPAR` into `STRING`. It also **returns** the `LENGTH` of the string (or zero if the parameter is not present).

[Previous][Next] [Up][Down]

5.4 The `graphserv` library

The `graphserv` library groups low level graphics routines, which fall in three specific categories :

- communication channel management routines, used to setup and control the couple of channels used to communciate between a graphics client and its server.
- a single piece of code f2x, represeneing the Fortran callabe Xlib interface used by the X window graphics server
- low level graphics proper (the `y_*.f` family) which implement the graphics primitives described in doc TBD

Use also the quick alphabetic index here below to locate the routine of interest.

- **blkpipecommon**

Library	graphserv	Fortran code
Calling sequence	EXTERNAL BLKPIPECOMMON	

This `BLOCK DATA` routine is called implicitly by the isregistered call to initialize the PIPECOMMON common block used to keep track of the list of registered graphics servers.

- ## connectserver

Library	graphserv	Fortran code
Calling sequence	`CALL CONNECTSERVER(CODE,N,lu1,lu2,IERR)`	
Arguments	`CHARACTER*(*)`	`CODE`
	`INTEGER`	`N`
	`INTEGER`	`LU1,LU2`
	`INTEGER`	`IERR`

This call establishes the connection to the `N`-th instance of the graphics server of type `CODE` (chosen among `'XW'|'BW'|'CP'|'C2'`), verifying the correct registration and the existence of the communication channels which are open on a couple of logical units `LU1,LU2` (there are always an input and an output channel per server !) using the VOS z_channel routine.

The returned error code `IERR` can be 0 in case of successful opening or 2 (server not registered), 3 (communication channels not set up correctly) or 5 (VOS error opening communication channels).

- ## deregister

Library	graphserv	Fortran code
Calling sequence	CALL DEREGISTER(CODE,N)	
Arguments	`CHARACTER*(*)`	`CODE`
	`INTEGER`	`N`

This call removes the registration entry of the `N`-th instance of the graphics server of type `CODE` (chosen among `'XW'|'BW'|'CP'|'C2'`) from the PIPECOMMON common block and from the XAS environment

---

- ## f2x

Library	graphserv	C code

Calling sequence	`EXTERNAL F2X`
	`...`
	`F2C_`*`element`*`=value`
	`F2C_`*`element`*`=value`
	`...`
	`CALL F2X_`*`routine`*	

Arguments	none	

This piece of C code is the Fortran-callable Xlib interface used only by the X window server `xwserver`. All routines are (from the Fortran point of view) argumentless entry points in the same C code file.
All communication with the caller occurs via a `struct` (actually a couple of them to preserve the usual separation between numeric and character data) seen by the Fortran program as a COMMON block : the caller sets specific elements, call one of the f2x routines (without arguments) and reads any return value in other elements of the COMMON.
All C routines in the f2x.c file share a local set of variables among themselves.
The entry points are the following (for further details see the code) :

- f2x : no-op (for relocation only)
- f2x_init : open display, create and map window, set up graphic context etc.
- f2x_winsize : i/f to XGetGeometry
- f2x_clear : i/f to XClearWindow
- f2x_clearview : i/f to XClearArea
- f2x_clip : i/f to XSetClipRectangles
- f2x_clipoff : i/f to XSetClipMask
- f2x_draw : i/f to XDrawLine
- f2x_polyl : i/f to XDrawLines
- f2x_polyg : i/f to XFillPolygon
- f2x_wrima : i/f to XAddPixel and XPutImage
- f2x_text : i/f to XDrawString
- f2x_colour : i/f to XSetForeground
- f2x_bkg : i/f to XSetBackground
- f2x_attrib : i/f to XSetLineAttributes
- f2x_font : i/f to XSetFont
- f2x_lutquery : i/f to XQueryColors
- f2x_lutalloc : i/f to XAllocColorCells and XStoreColors
- f2x_lutchange : i/f to XStoreColors
- f2x_lutdummy : internal i/f
- f2x_curson : i/f to XWindowEvent etc. (cursor management)
- f2x_disconn : send XSync before disconnecting
- f2x_end : i/f to XCloseDisplay

- **isregistered**

Library	graphserv	Fortran code
Calling sequence	CALL ISREGISTERED(code,n,ok) IF (OK) THEN ...	
Arguments	CHARACTER*(*)	CODE
	INTEGER	N
	LOGICAL	OK

This call verifies a registration entry for the N-th instance of the graphics server of type CODE (chosen among 'XW'|'BW'|'CP'|'C2') is present in the XAS environment, and returns a flag OK=.TRUE. if the server is registered.
Side effects: The environment is loaded in the PIPECOMMON common block while at the same time the index of the server and the next free slot for a server are computed.

- **register**

Library	graphserv	Fortran code
Calling sequence	CALL REGISTER(CODE,N)	
Arguments	CHARACTER*(*)	CODE
	INTEGER	N

This call records a registration entry of the N-th instance of the graphics server of type CODE (chosen among 'XW'|'BW'|'CP'|'C2') in the next free slot of the PIPECOMMON common block and in the XAS environment

- **y_clear_viewport**

Library	graphserv	Fortran code
Calling sequence	CALL Y_CLEAR_VIEWPORT(LUS)	
Arguments	INTEGER	LUS(2)

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 8 which causes the graphics viewport to be cleared.

- **y_closeplot**

Library	graphserv	Fortran code
Calling sequence	CALL Y_CLOSEPLOT(LUS)	
Arguments	INTEGER	LUS(2)

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 0 which notifies the server of the disconnection of the client. Note that the communication channel is not closed (that is responsibility of the server, not of the client !).

- **y_colour**

Library	graphserv	Fortran code
Calling sequence	CALL Y_COLOUR(LUS,ICOL)	
Arguments	INTEGER	LUS(2)
	INTEGER	ICOL

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 105 with operand the colour ICOL, which sets the current "pen colour" to the appropriate value (colours 0-7 are the standard simple colours, while a negative ICOL points to location ABS(ICOL) of the current colour lookup table.

- **y_coordinates**

Library	graphserv	Fortran code
Calling sequence	CALL Y_COORDINATES(LUS,icode)	
Arguments	INTEGER	LUS(2)
	INTEGER	ICODE

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 103 with operand the colour ICODE, which tells the server to interpret all further coordinates in raw device coordinates (ICODE=0), normalized device coordinates (NDC, 0.0-1.0, ICODE=1), world coordinates (ICODE=2), cm or inch (respectively ICODE=3 or 4)

- **y_draw**

Library	graphserv	Fortran code
Calling sequence	CALL Y_DRAW(LUS,X,Y)	
Arguments	INTEGER	LUS(2)
	REAL	X,Y

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 3 with operands the coordinates of a point in the current coordinate system X,Y, which tells the server to draw a line from the current point to X,Y.

- **y_fill**

Library	graphserv	Fortran code
Calling sequence	CALL Y_FILL(LUS,N,X,Y)	
Arguments	INTEGER	LUS(2)
	INTEGER	N
	REAL	X(N) ,Y(N)

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the polyfill opcode 6 with operands the number of points N, and an array of coordinates in the current coordinate system X,Y, which tells the server to fill (with the current pen colour) the polygon enclosed by points X,Y.

- **y_get_cursor**

Library	graphserv	Fortran code
Calling sequence	CALL Y_GET_CURSOR(LUS,X,Y,A)	
Arguments	INTEGER	LUS(2)
	REAL	X,Y
	CHARACTER	A

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 11 (which causes the server to read the current cursor position at a key or button press) and reads from the server output channel the coordinate of the point in the current coordinate system X,Y, and the character of the pressed key A (mouse buttons currently return '?').

- **y_lines**

Library	graphserv	Fortran code
Calling sequence	CALL Y_LINES(LUS,N,X,Y)	
Arguments	INTEGER	LUS(2)
	INTEGER	N
	REAL	X(N) ,Y(N)

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the polyline opcode 4 with operands the number of points N, and an array of coordinates in the current coordinate system X,Y, which tells the server to draw (with the current pen colour) a (poly)line connecting points X,Y.

- **y_move**

Library	graphserv	Fortran code
Calling sequence	CALL Y_MOVE(LUS,X,Y)	
Arguments	INTEGER	LUS(2)
	REAL	X,Y

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 2 with operands the coordinates of a point in the current coordinate system X,Y, which tells the server to move the current point to X,Y.

- **y_openplot**

Library	graphserv	Fortran code

	CHARACTER*(*)	CODE
	INTEGER	N

This routine is called by a graphics client to establish a plotting connection (via connectserver) to the N-th instance of the graphics server of type CODE (chosen among 'XW'\|'BW'\|'CP'\|'C2' or ' ' in which case the default server xw1 is used). The couple of logical units LUS, associated to the communication channels, are used in all subsequent graphics calls.

- ## y_page

Library	graphserv	Fortran code
Calling sequence	CALLY_PAGE(LUS)	
Arguments	INTEGER	LUS(2)

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 1 , which causes the server to clear the entire page or screen.

- ## y_readlut

Library	graphserv	Fortran code
Calling sequence	CALL Y_READLUT(LUS,START,NSTEP,ISIGN,R,G,B)	
Arguments	INTEGER	LUS(2)
	INTEGER	START
	INTEGER	NSTEP
	INTEGER	ISIGN
	REAL	R(NSTEP), G(NSTEP),G(NSTEP)

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 13 (which causes the server to read the current colour lookup table) and reads from the server output channel the red, green and blue components (0.0-1.0) R,G,B of NSTEP locations of the lookup table, starting at location START. The starting location can be relative (ISIGN=+1) to the beginning of the colour table loaded with y_writelut, or absolute (ISIGN=-1), referred to the origin of the system colour table (the XAS colour table loaded with y_writelut occupies only a portion of the typical X colour table).

- ## y_scale

Library	graphserv	Fortran code
Calling sequence	CALL Y_SCALE(LUS,X,Y)	
Arguments	INTEGER	LUS(2)
	CHARACTER	X,Y ('LIN'|'LOG')

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 104 with a couple of 0 or 1 operands, which tells the server to plot all further data converting to linear or logarithmic scale as specified for the X,Y axes by the appropriate 'LIN' or 'LOG' codes.

- ## y_text

Library	graphserv	Fortran code
Calling sequence	CALL Y_TEXT(LUS,STRING,IFONT)	
Arguments	INTEGER	LUS(2)
	CHARACTER*(*)	STRING
INTEGER	IFONT	

This routine sends to the graphics server at the other end of the communication channels attached to logical units LUS the opcode 110 with operand a font number IFONT, followed by opcode 7 with operands the length of a string STRING, and the character string itself. This causes the server to write at the current position the text string in the wished font.
Font indexes are server dependent and are controlled by entries in a server specific font listing file

---

- ## y_viewport

Library	graphserv	Fortran code
Calling sequence	CALL Y_VIEWPORT(LUS,X1,X2,Y1,Y2)	
Arguments	INTEGER	LUS(2)
	REAL	X1,X2,Y1,Y2

This routine sends to the graphics server at the other end of the communication channels attached to logical units `LUS` the opcode 101 with operands the coordinates of two points in the current coordinate system `X1,Y1` and `X2,Y2`, which are intepreted as the corners of the new viewport.

- ## y_width

Library	graphserv	Fortran code
Calling sequence	`CALL Y_WIDTH(LUS,WIDTH)`	
Arguments	INTEGER	LUS(2)
	REAL	WIDTH

This routine sends to the graphics server at the other end of the communication channels attached to logical units `LUS` the opcode 106 with operand the line width `WIDTH`, which tells the server to draw all further lines in the (server dependent) width.

- ## y_window

Library	graphserv	Fortran code
Calling sequence	`CALL Y_WINDOW(LUS,X1,X2,Y1,Y2)`	
Arguments	INTEGER	LUS(2)
	REAL	X1,X2,Y1,Y2

This routine sends to the graphics server at the other end of the communication channels attached to logical units `LUS` the opcode 102 with operands the coordinates of two points in the current coordinate system `X1,Y1` and `X2,Y2`, which are intepreted as the corners of the new plotting window, then sends an opcode 103 with operand 2, to force the coordinate system into world coordinates.

- ## y_write_image

Library	graphserv	Fortran code
Calling sequence	`CALL Y_WRITE_IMAGE(LUS,DATA,ILEN)`	
Arguments	INTEGER	LUS(2)
	CHARACTER*(ILEN)	DATA
	INTEGER	ILEN

This routine sends to the graphics server at the other end of the communication channels attached to logical units `LUS` the opcode 9 with operands the number of bytes `ILEN` (max 4096) and a byte string and `DATA`, which cause the server to display one image line. The image `DATA` shall be prescaled in values ranging 0-255 (or less) pointing to colours in the current colour lookup table.

- ## y_writelut

Library	graphserv	Fortran code
Calling sequence	`CALL Y_WRITELUT(LUS,START,NSTEP,R,G,B)`	
Arguments	INTEGER	LUS(2)
	INTEGER	START
	INTEGER	NSTEP
	REAL	R(NSTEP), G(NSTEP),G(NSTEP)

This routine sends to the graphics server at the other end of the communication channels attached to logical units `LUS` the opcode 10 (which causes the server to write values into the current colour lookup table) and loads the red, green and blue components (0.0-1.0) `R,G,B` into `NSTEP` locations of the lookup table (LUT), starting at location `START`. The start and step parameters are interpreted by the server as follows :

- `START=0 NSTEP.NE.0`
  allocates (or reallocates) a new contiguous LUT segment wherever the X server likes to, or redefines entire LUT for the Postscript servers.
- `START.NE.0 NSTEP.NE.0`
  redefines the content of part of the current LUT, i.e. loads `NSTEP` values after relative `START`.
- `START.NE.0 NSTEP=0`
  assigns explicitly location `START` in the X server colour table as start for the XAS LUT (unused for Postscript servers)
- `START=0 NSTEP=0` (query call),
  writes on terminal the current LUT (relative location 0) start in the X server colour table (unused for Postscript servers)

[Previous][Next] [Up][Down]

5.5 The `xasgraph` library

The `xasgraph` library groups high level graphics calls of miscellaneous nature (they are layered upon the low level calls in the graphserv library) .

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

- ## annotate_new

Library	xasgraph	Fortran code
Calling sequence	`CALL ANNOTATE_NEW(LUS,ANNOTATE,BUFFER,BUFFER2,NAME)`	
Arguments	`INTEGER`	`LUS(2)`
	`CHARACTER`	`ANNOTATE ('N'|'S'|'L')`
	`CHARACTER*(*)`	`BUFFER,BUFFER2`
	`CHARACTER*(*)`	`NAME`

This high level routine uses the logical units `LUS` pointing to the communication channels of a plotting server established by y_openplot to set the viewport for standard annotations (which include the XAS filename `NAME` and other information derived from its header) and display the annotations.
The annotation style (none, short or long) is controlled by the `ANNOTATE` flag (respectively `('N'|'S'|'L')`.
The two strings `BUFFER,BUFFER2` are used internally as work areas.

- ## annotate_old

Library	xasgraph	Fortran code
Calling sequence	`CALL ANNOTATE_OLD(LUS,ANNOTATE,BUFFER,NAME)`	
Arguments	see previous routine	

A variant of annotate_new

- ## check_overtrace

Library	xasgraph	Fortran code
Calling sequence	`CHECK_OVERTRACE(CLEAR)`	
Arguments	`LOGICAL`	`CLEAR`

This routine, called at the beginning of a graphical client, tests the transient XAS environment variable overwrite, unsets it, and returns a logical flag `CLEAR` (`.TRUE.` if the current plot shall overtrace the preexisting one, `.FALSE.` otherwise).

- ## df_axes

Library	xasgraph	Fortran code
Calling sequence	`CALL DF_AXES(LUS,XLINLOG,YLINLOG, XUNIT,YUNIT, XFORMAT,YFORMAT, BUFFER)`	
Arguments	`INTEGER`	`LUS(2)`
	`CHARACTER*3`	`XLINLOG,YLINLOG ('LIN'|'LOG')`
	`CHARACTER*(*)`	`XUNIT,YUNIT`
	`CHARACTER*(*)`	`XFORMAT,YFORMAT`
	`CHARACTER*(*)`	`BUFFER`

This routine plots an axis frame (using the logical units `LUS` pointing to the communication channels of a plotting server established by y_openplot) annotating them using the lb_* routines described below. The type of axis (linear or logarithmic) `XLINLOG,YLINLOG` and the format of the numeric labels `XFORMAT,YFORMAT` are passed to such routines.
The caption of the X and Y axes are taken instead from an appropriate keyword header in a XAS table file : i.e. if one is plotting the 4-th vs the 3-rd column of a table, and therefore the X column is column 3 of the table containing energy in keV, one puts `XUNIT='TUNIT3'` where the `TUNIT3` keyword contains `'keV'`. `BUFFER` is a work area supplied by the caller and used to construct labels.

- ## df_pen_colours

Library	xasgraph	Fortran code
Calling sequence	CALL DF_PEN_COLOURS(LUS,CLEAR)	
Arguments	INTEGER	LUS(2)
	LOGICAL	CLEAR

This routine sets the foreground (and background, not yet implemented) colours for the plotting server connected by y_openplot) to the logical units LUS) according to a family of XAS environment variables which are read in common PENCOMMON (axispen, bkgpen, datapen, CHARACTERANNOTATE ('N'|'S'|'L') textpen and by default any missing one to pen and by default if missing to 1).
In addition if the CLEAR flag has been set by check_overtrace, it increments the overcount environment variables and cycles the text and data pens among the 8 fundamental colours.

- ## df_viewport

Library	xasgraph	Fortran code
Calling sequence	CALL DF_VIEWPORT(LUS,BUFFER,ANNOTATE)	
Arguments	INTEGER	LUS(2)
	CHARACTER*(*)	BUFFER
	CHARACTER	ANNOTATE ('N'|'S'|'L')

This routine sets the default viewport for the plotting server connected by y_openplot) to the logical units LUS, according to the ANNOTATE flag set by annotate_new (this takes account of space around the data frame used for annotations). It uses a default viewport, unless specifically set by the user in XAS environment variable viewport, and stores the information back in the environment. The viewport is changed only if the annotation style has been changed since last plot (this is also kept track of in the environment)
BUFFER is a work area supplied by the caller and used internally.

- ## df_window

Library	xasgraph	Fortran code
Calling sequence	CALL DF_WINDOW(LUS,CLEAR,REPLOT, BUFFER, XLINLOG,YLINLOG, XWL,XWU ,YWL,YWU, XWL1,YWL1)	
Arguments	INTEGER	LUS(2)
	LOGICAL	CLEAR
	LOGICAL	REPLOT
	CHARACTER*(*)	BUFFER
	CHARACTER*3	XLINLOG,YLINLOG ('LIN'|'LOG')
	REAL	XWL,XWU ,YWL,YWU
	REAL	XWL1,YWL1

This routine associates a plotting window to the the default viewport set by df_viewport for the plotting server connected by y_openplot to the logical units LUS.
It **returns** the linear or logarithmic setting XLINLOG,YLINLOG for the axis scales retrieving it from the XAS environment.
The setting of the axes is done only for new frames (otherwise they are retrieved from the XAS environment), i.e. if the CLEAR flag set by check_overtrace or the REPLOT flag are .TRUE..
The coordinates of the lower left and upper right corners XWL,YWL XWU,YWU of the plotting windows are treated as suggestions and rounded to nices values by the nice_* calls as instructed by XAS environment.
The actual values are stored as side effect in PENCOMMON.
Note that for log scales the lower bounds of the axes cannot be smaller than the positive safety values in XWL1,YWL1.
BUFFER is a work area supplied by the caller and used internally.

- ## get_datastyle

Library	xasgraph	Fortran code
Calling sequence	CALL GET_DATASTYLE(ISTYLE)	
Arguments	INTEGER	ISTYLE

This routine tests the datastyle XAS environment variable and **returns** a numeric flag indicating the plotting style :

1. for the Histogram style
2. for the Error bar style
3. for the Solid style
4. for the Marker style (unimplemented and replace by Histogram style)

- ## lb_axis

A family of three axis plotting and annotation routines.

Library	xasgraph	Fortran code
Calling sequence	CALL LB_AXIS(XY,ARG2,ARG3, LINLOG, ARG5,ARG6,ARG7 ,ARGX,NTIC)	
Arguments	not documented online	

This routine draws an axis with its annotations. It is derived from a pre-existing plotting package called "The Labeller" which is documented on paper separately.

- ## lb_number

Library	xasgraph	Fortran code
Calling sequence	CALL LB_NUMBER(LUS,ARG1,ARG2,ARG3)	
Arguments	INTEGER	LUS(2)
	CHARACTER*(*)	ARG1
	CHARACTER*(*)	ARG2 ('CENTRED')
	CHARACTER*(*)	ARG3 ('NORMAL'|'ROTATED')

This routine (derived from the same pre-existing plotting package mentioned for lb_axis) plots the tics values for axes.
ARG2 controls the alignment of the number label with respect to the tic coordinate (only 'CENTRED' out of the original list of possible values 'PROTECTED'| 'CENTRED'| 'LEFT'| 'RIGHT'| 'GRADUAL' is supported).
ARG3 controls the orientation of the number label, which is either 'NORMAL' or 'ROTATED' by 90 degrees.
ARG1 is the format of the numeric label. Integer and floating point Fortran style formats ('In', 'Fw.d') are supported together with an 'Hn' format used to obtain hh:mm:ss hour notation, an 'EXP' and a 'LOG' formats (the former intended for 10^{n} notations, while the second reports only the exponent n and is typically used for log scales with many decades).

- ## lb_tics

Library	xasgraph	Fortran code
Calling sequence	CALL LB_TICS(LUS,ARG1)	
Arguments	INTEGER	LUS(2)
	REAL	ARG1

This routine (derived from the same pre-existing plotting package mentioned for lb_axis) plots the tics at appropriate places, where ARG1 is the tick height (or length) in NDC (Normalized Device Coordinates, 0.0-1.0, i.e. as a fraction of viewport size).

- ## nice_axes

A family of three routines to set up axis tic labelling.

Library	xasgraph	Fortran code
Calling sequence	CALL NICE_AXES(XYMIN,XYMAX,XYLOW,XYUPP)	
Arguments	REAL	XYMIN,XYMAX
	REAL	XYLOW,XYUPP

This routine, passed the suggested extrema of an axis XYMIN,XYMAX **returns** nice rounded extrema XYLOW,XYUPP. Rounding is done down or up to the nearest power of ten (e.g. 2.7 is rounded down to 2 or up to 3, while -440 is rounded down to -500 or up to -400).

- ## nicer_lin_axes

Library	xasgraph	Fortran code
Calling sequence	CALL NICER_LIN_AXES(XYMIN,XYMAX,XYLOW,XYUPP,NTIC)	
Arguments	REAL	XYMIN,XYMAX
	REAL	XYLOW,XYUPP
	INTEGER	NTIC

An improved version of nice_axes which does the rounding so that the values of all tic labels looks nice, when the axis is divided into NTIC parts.

- ## nicer_log_axes

Library	xasgraph	Fortran code
Calling sequence	CALL NICER_LOG_AXES(XYMIN,XYMAX,XYLOW,XYUPP)	
Arguments	REAL	XYMIN,XYMAX
	REAL	XYLOW,XYUPP

An improved version of nice_axes which does the rounding so that the values of tic labels looks nice, when the axis is logarithmic.

- ## plot_xxy_bar

A family of routines to plot Y vs X data in a number of standard ways. All routines as usual plot on the plotting server connected by y_openplot to the logical units LUS.

Library	xasgraph	Fortran code
Calling sequence	CALL PLOT_XXY_BAR(LUS,LOWER,UPPER, DATA,ERROR, NCHAN, X,Y)	
Arguments	INTEGER	LUS(2)
	REAL	LOWER(NCHAN), UPPER(NCHAN)
	REAL	DATA(NCHAN), ERROR(NCHAN)
	INTEGER	NCHAN
	REAL	X(2),Y(2)

This routine plots an array of NCHAN error bars (or upper limits if the ulimit environment variable is set). The DATA and ERROR arrays are the Y data and their errors. The LOWER and UPPER arrays are the extrema of the X error bars.
The X and Y arrays (of size 2) are work areas used internally to construct the points to be plotted at the ends of the bars.

- ## plot_xxy_histo

Library	xasgraph	Fortran code
Calling sequence	CALL PLOT_XXY_HISTO(LUS,LOWER,UPPER, DATA, NCHAN, X,Y)	
Arguments	INTEGER	LUS(2)
	REAL	LOWER(NCHAN), UPPER(NCHAN)
	REAL	DATA(NCHAN),
	INTEGER	NCHAN
	REAL	X(2*NCHAN),Y(2*NCHAN)

This routine plots an array of NCHAN DATA Y values in histogram style (i.e. connecting horizontal bars at the given Y height, where the LOWER and UPPER arrays are the extrema of the bars in X.
The X and Y arrays (of double size) are work areas used internally to construct all the points to be plotted. The y_lines polyline primitive is used, plotting such data in chunks of 512 points (because of a limitation in the server communication channel buffer) transparently to the user.
The routine interrupts the plot if a gap (of more than 1% of the previous bin size) is found between the LOWER end of a bin and the UPPER end of the previous bin, and resumes at the next bin (results can be surprising when plotting very sparse data !)

- ## plot_xxy_join

Library	xasgraph	Fortran code
Calling sequence	CALL PLOT_XXY_JOIN(LUS,LOWER,UPPER, DATA, NCHAN, X,Y)	
Arguments	INTEGER	LUS(2)
	REAL	LOWER(NCHAN), UPPER(NCHAN)
	REAL	DATA(NCHAN),
	INTEGER	NCHAN
	REAL	X(NCHAN),Y(NCHAN)

This routine plots (connecting each point) an array of NCHAN DATA Y values vs X where the X coordinates are the mean between the LOWER and UPPER arrays.
The X and Y arrays (of same size as the data) are work areas used internally to construct all the points to be plotted. The y_lines polyline primitive is used, plotting such data in chunks of 512 points (because of a limitation in the server communication channel buffer) transparently to the user.
The routine interrupts the solid line if a gap (of more than 1% of the previous bin size) is found between the LOWER end of a bin and the UPPER end of the previous bin.

- ## plot_xy_join

Library	xasgraph	Fortran code
Calling sequence	CALL PLOT_XY_JOIN(LUS,XAX,DATA ,NCHAN, X,Y)	
Arguments	INTEGER	LUS(2)
	REAL	XAX(NCHAN)
	REAL	DATA(NCHAN),
	INTEGER	NCHAN
	REAL	X(NCHAN),Y(NCHAN)

This routine plots (connecting each point) an array of NCHAN DATA Y values vs the XAX X values.
The X and Y arrays (of same size as the data) are work areas used internally to construct all the points to be plotted. The y_lines polyline primitive is used, plotting such data in chunks of 512 points (because of a limitation in the server communication channel buffer) transparently to the user.

[Previous][Next] [Up][Down]

5.6 The fotlib library

The fotlib library groups mission-dependent (but instrument independent) routines specific for access to SAX FOT telemetry data files.

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

- ## add_end

Library	fotlib	Fortran code
Calling sequence	CALL ADD_END(ipass)	
Arguments	INTEGER	IPASS

Category: specific of fotfile program
This routine adds to an external shell script file the commands necessary to run the next (IPASS-th, 1,2 or 3) pass of fotfile.

- ## add_file

Library	fotlib	Fortran code
Calling sequence	CALL ADD_FILE(file,blksize,recl,type)	
Arguments	CHARACTER*(*)	FILE
	INTEGER	BLKSIZE, RECL
	CHARACTER*3	TYPE ('ASC'|'BIN')

Category: specific of fotfile program
This routine adds to an external shell script file the commands necessary to read from tape a file called FILE unblocking it from tape block size BLKSIZE to disk record length RECL according to the rules for ASCII or binary decoding as specified by TYPE.
The system (and site) dependent template commands are read from a programming support file located in the local directory and edited replacing the tokens with the subroutine arguments, using edit_cmd

- ## add_init

Library	fotlib	Fortran code
Calling sequence	CALL ADD_INIT	

Category: specific of fotfile program
This routine adds to an external shell script file the commands necessary to begin the operations.

- ## add_rew_tape

Library	fotlib	Fortran code
Calling sequence	CALL ADD_REW_TAPE	

Category: specific of fotfile program
This routine adds to an external shell script file the command necessary to rewind the tape. The system (and site) dependent template command is read from a programming support file located in the local directory and edited replacing the tokens with the tape drive name (present in a COMMON block) using edit_cmd

- ## add_skip_tape

Library	fotlib	Fortran code
Calling sequence	CALL ADD_SKIP_TAPE(N)	
Arguments	INTEGER	N

Category: specific of fotfile program
This routine adds to an external shell script file the command necessary to skip forward N tape files. The system (and site) dependent template command is read from a programming support file located in the local directory and edited replacing the tokens with the routine argument using edit_cmd

---

- ## check_packet

Library	fotlib	Fortran code
Calling sequence	CALL CHECK_PACKET(PACKET,DATATYPE,N)	
Arguments	CHARACTER*(*)	PACKET
	CHARACTER	DATATYPE ('I'|'S'|'P'|'T'|'HK')
	INTEGER	N

Returns the number N of packet families listed in the instrument directory and eligible for the requested accumulation DATATYPE (where the codes are for Images, Spectra, Photon files, Time profiles or HK time profiles). The PACKET argument is unused, included for compatibility with sax_which_data

---

- ## correct

Library	fotlib	Fortran code
Calling sequence	CALL CORRECT	

A case statement calling the appropriate event correction routine for MECS, LECS, PDS or HPGSPC.

- ## edit_cmd

Library	fotlib	Fortran code
Calling sequence	CALL EDIT_CMD(STRING,OLD,NEW)	
Arguments	CHARACTER*(*)	STRING
	CHARACTER*(*)	OLD,NEW

A general STRING editing routine which replaces the first occurrence of a string token OLD with the string value NEW.
Used in particular by tape command generation routines for the fotfile program.

- ## exposure_b1s1

Library	fotlib	Fortran code
Calling sequence	CALL EXPOSURE_B1S1(ICOUNTER, ISTIME,ITIMSIZ, RECORD,CONVERT, PREVIOUS_TIMEHI, ENDWINDOW,ENDACCUM, NEWOBS,TCYCLE)	
Arguments	undocumented	

A service routine called by sax_acc_b1s1_y to update the contribution to the live exposure time of the current packet (the value is stored in TIMECOMMON). See code for further details.

- ## exposure_b1s3

Library	fotlib	Fortran code
Calling sequence	CALL EXPOSURE_B1S3(ICOUNTER, ISTIME,ITIMSIZ, RECORD,CONVERT, PREVIOUS_TIMEHI,ENDWINDOW,ENDACCUM, NEWOBS,TCYCLE, recycled)	
Arguments	undocumented	

A service routine called by sax_acc_b1s3_y to update the contribution to the live exposure time of the current packet (the value is stored in TIMECOMMON) with the same logics of the previous routine. See code for further details.

- ## get_start_end

Library	fotlib	Fortran code
Calling sequence	CALL GET_START_END(TMIN,TMAX)	
Arguments	DOUBLE PRECISION	TMIN,TMAX

Category: accumulation support

This routine **returns** the default start and end times for the accumulation, i.e. the start time of the first observation and the end time of the last observation in the current chain (as defined in the XAS environment), read from the instrument directory.
`TMIN`, `TMAX` are in seconds (or with the resolution specified by the time constant setup)
Side effects: information about start and end times are also stored in the OPCOMMON and TIMECOMMON blocks.

- ## init_correct

Library	fotlib	Fortran code
Calling sequence	`CALL INIT_CORRECT(STUFF)`	
Arguments	`CHARACTER*(*)`	`STUFF(*)`

Essentially a case statement calling the appropriate event correction initialization routine for MECS, LECS, PDS or HPGSPC, in the case the environment requires event correction.
The array `STUFF` lists the names of all event quantities which are involved in the correction.

- ## instr_keywords

Library	fotlib	Fortran code
Calling sequence	`CALL INSTR_KEYWORDS(PRODUCT)`	
Arguments	`CHARACTER`	`PRODUCT ('I'|'S'|'P'|'T'|'H'|'M')`

A case statement calling the appropriate routine which adds instrument specific header keywords for MECS, LECS, PDS or HPGSPC. The `PRODUCT` code is used only by the PDS call, which has different sets of keywords for Images, Spectra, Photon files, Time profiles, HK time profiles or Matrices.

- ## lintomm

Library	fotlib	Fortran code
Calling sequence	`CALL LINTOMM(X,Y)`	
Arguments	`DOUBLE PRECISION`	`X,Y`

Category: attitude
Converts **in place** the coordinates `X,Y` from linearized pixels to mm on the focal plane. Although the routine is potentially general, it is implemented only for MECS since it needs to load the current pixel size.

- ## mmtopix

Library	fotlib	Fortran code
Calling sequence	`CALL MMTOPIX(X,Y)`	
Arguments	`DOUBLE PRECISION`	`X,Y`

Category: attitude
Converts **in place** the coordinates `X,Y` from mm on the focal plane to linearized pixels. Although the routine is potentially general, it is implemented only for MECS since it needs to load the current pixel size.

- ## rearrange_instrec

Library	fotlib	Fortran code
Calling sequence	`CALL REARRANGE_INSTREC(INSTREC)`	
Arguments	`CHARACTER*133`	`INSTREC`

A service routine (at present used only by get_start_end and pds_keywords) to reformat an instrument directory record enclosing hex-coded values in quotes, so that it can be read with list-directed i/o.

- ## sax_acc_b1sn_x

The sax_acc_b1sn_x family of routines are the packet handlers for direct mode telemetry packets (basic type 1, secondary type n). The x code equal to y indicates packets with fields reformatted to span always an integer number of bytes (as are all FOT telemetry packets), while an x code equal to i indicates fields which can use any number of bits (not necessarily multiple of 8), which was the case of raw telemetry used during ground calibrations. The latter routines are not publicly distributed (but replaced with "return end" routines) and not documented.

All these routines have a single argument, the name of an `INCREMENT_ROUTINE` which must be called to process a single event (or packet, or logical portion thereof) in a way specified by each accumulation program (i.e. such routine is in the main program source file and not a library routine).

These routines take care of all byte swapping (as dictated by the telemetry description files and the characteristics of the current operating system), of calling event corrections if required, of advancing from the telemetry file of one observation to the next in the chain, and handle time windows.

- ## sax_acc_b1s1_i

Library	fotlib	Fortran code
Calling sequence	SAX_ACC_B1S1_I(INCREMENT_ROUTINE)	

Bit (ground calibration raw data) version of the next routine.

- ## sax_acc_b1s1_y

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_B1S1_Y(INCREMENT_ROUTINE)	
Arguments	EXTERNAL	INCREMENT_ROUTINE

This routine deals with basic type 1, secondary type 1 i.e. the MECS, PDS and HPGSPC direct mode packets. It computes the packet exposure time via exposure_b1s1 and decodes the events calling the INCREMENT_ROUTINE once per event.

- ## sax_acc_b1s2_i

Library	fotlib	Fortran code
Calling sequence	SAX_ACC_B1S2_I(INCREMENT_ROUTINE)	

Bit (dummy) version of the next routine.

- ## sax_acc_b1s2_y

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_B1S2_Y(INCREMENT_ROUTINE)	

Intended to deal with basic type 1, secondary type 2 i.e. the WFC direct mode packets, is unfinished and untested.

- ## sax_acc_b1s3_y

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_B1S3_Y(INCREMENT_ROUTINE)	

This routine deals with basic type 1, secondary type 3 i.e. the LECS direct mode packets. It includes code for correcting some of the known errors in such packet times (either present ab origine or introduced at FOT production level). It computes the packet exposure time via exposure_b1s3 and decodes the events calling the INCREMENT_ROUTINE once per event.

- ## sax_acc_b2s1_y

The sax_acc_b2sn_x family of routines are the packet handlers for indirect mode telemetry packets (basic type 2, secondary type n). These routines are structured similarly to the sax_acc_b1sn_x ones.

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_B2S1_Y(INCREMENT_ROUTINE)	

This routine deals with basic type 2, secondary type 1 i.e. the MECS, PDS and HPGSPC indirect mode packets. Such packet include one or more spectra : the INCREMENT_ROUTINE is called once per spectrum. These routines have been poorly tested since indirect packets are hardly used.

- ## sax_acc_b3s1

The sax_acc_b3sn_x family of routines are the packet handlers for HouseKeeping mode telemetry packets (basic type 3, secondary type n). These routines are structured similarly to the sax_acc_b1sn_x ones.

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_B3S1(INCREMENT_ROUTINE)	

Unused for flight data. Deals with ground data (Laben Block Transfer Bus instrument HK).

- ## sax_acc_b3s2

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_B3S2(INCREMENT_ROUTINE)	

Unused for flight data. Deals with ground data (Laben/Alenia Interrrogation/Response Bus instrument and spacecraft HK).

- **sax_acc_b3s3**

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_ACC_B3S3(INCREMENT_ROUTINE)`	

This routine deals with basic type 2, secondary type 3 i.e. the MECS, PDS and HPGSPC ENGineering mode packets (containing ratemeters). It calls the `INCREMENT_ROUTINE` once per ratemeter sample.

- **sax_acc_b3s4**

Library	fotlib	Fortran code
Calling sequence	`SAX_ACC_B3S4(INCREMENT_ROUTINE)`	

This routine deals with basic type 2, secondary type 4 i.e. the MECS, PDS and HPGSPC HKD mode packets (containing instrument HK parameters from Virtual Channel 1). It calls the `INCREMENT_ROUTINE` once per parameter sample.

- **sax_acc_b3s5**

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_ACC_B3S5(INCREMENT_ROUTINE)`	

This routine deals with basic type 2, secondary type 5 i.e. spacecraft HKD mode packets (containing spacecraft HK parameters). It calls the `INCREMENT_ROUTINE` once per parameter sample.

- **sax_acc_b3s6**

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_ACC_B3S6(INCREMENT_ROUTINE)`	

This routine deals with basic type 2, secondary type 6 i.e. pseudo-spacecraft ASCII data, namely the attitude and ephemeris files. It calls the `INCREMENT_ROUTINE` once per attitude or ephemeris record. An internal hidden routine reformats "new" attitude records (which are not compliant with the original ICD) so that they can be read with list-directed i/o as old were.

- **sax_acc_bt_1**

The family of `sax_acc_bt_j` are trivial dispatchers handling a request for packets of basic type j and calling the appropriate "secondary type" routine sax_acc_bjsn_x. They all share a similar calling sequence

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_ACC_BT_1(IST,UNITS,INCREMENT_ROUTINE)`	

is the dispatcher for direct mode data (basic type 1).

- **sax_acc_bt_2**

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_ACC_BT_2(IST,UNITS,INCREMENT_ROUTINE)`	

is the dispatcher for indirect mode data (basic type 2).

- **sax_acc_bt_3**

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_ACC_BT_3(IST,UNITS,INCREMENT_ROUTINE)`	
Arguments	`INTEGER`	`IST`
	`CHARACTER*(*)`	`UNITS ('BYTES'|'BITS')`
	`EXTERNAL`	`INCREMENT_ROUTINE`

is the dispatcher for HK mode data (basic type 3).

The secondary type `IST` and the unit for data `UNITS` are used to branch to the appropriate sax_acc_bjsn_x routine, to which the name of the `INCREMENT_ROUTINE` is passed straight on.

- **sax_acc_hkrange**

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_ACC_HKRANGE(NDIMENS, NFORMAT, XSTUFF, STUFF, PACKET)`	
Arguments	mostly unused, included for analogy with sax_acc_range	

Category: accumulation support

Handles the range dialogue for HK accumulations : in practice just asks and sets the start and end times and the binning time (multiple of HK sampling time) proposing appropriate defaults.

- ## sax_acc_loop

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_LOOP(INCREMENT_ROUTINE)	
Arguments	EXTERNAL	INCREMENT_ROUTINE

This routine includes the entire accumulation loop on all data packets in all observations of the observation chain. In fact this routine does nothing more than retrieving from the packetcap the basic information about the current selected type of telemetry packet, and dispatching the appropriate basic type sax_acc_bt_i dispatcher routine, which in turn calls the appropriate sax_acc_bjsn_x routine (to which the name of the INCREMENT_ROUTINE is passed straight on). It is in the sax_acc_bjsn_x routine that all loops are unrolled !

- ## sax_acc_open_sc_tlm

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_OPEN_SC_TLM(LU,PACKET)	
Arguments	INTEGER	LU
	CHARACTER*(*)	PACKET

Similar to the next routine, but opens spacecraft telemetry files (whose naming convention is slightly different, and may include ASCII files).

- ## sax_acc_open_tlm

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_OPEN_TLM(LU,PACKET)	
Arguments	INTEGER	LU
	CHARACTER*(*)	PACKET

Opens on logical unit LU the first telemetry file of the observation chain which is of type PACKET (resolving internally the file name).

- ## sax_acc_other_range

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_OTHER_RANGE(NDIMENS ,NFORMAT, XSTUFF, STUFF)	
Arguments	INTEGER	NDIMENS
	INTEGER	NFORMAT
	CHARACTER*(*)	XSTUFF(*),STUFF(*)

Category: accumulation support
Handles the range dialogue for uninteresting quantities in accumulations. The arguments are as for the corresponding call to sax_acc_range, which deals instead with interesting quantities. Essentially this routine constructs adequate defaults for the start and end value of each quantity (values out of range cause data rejection), and asks the user to confirm or change them (directly for all parameters except times, which are processed through asktime).

- ## sax_acc_preload

Library	fotlib	Fortran code
Calling sequence	CALL SAX_ACC_PRELOAD(NDIMENS, NFORMAT, XSTUFF, STUFF)	
Arguments	INTEGER	NDIMENS
	INTEGER	NFORMAT
	CHARACTER*(*)	XSTUFF(*)
	CHARACTER*(*)	STUFF(*)

Category: accumulation support
This routine retrieves from the packetcap of the current telemetry packet the basic information about each of the data quantities present in the file, inclusive of their names STUFF. The arguments are those listed for sax_acc_range, except that this call is the one which loads the values in STUFF

- ## sax_acc_range

Library	fotlib	Fortran code

Calling sequence	`CALL SAX_ACC_RANGE(NDIMENS, NFORMAT, XSTUFF, STUFF)`	
Arguments	`INTEGER`	`NDIMENS`
	`INTEGER`	`NFORMAT`
	`CHARACTER*(*)`	`XSTUFF(*)`
	`CHARACTER*(*)`	`STUFF(*)`

Category: accumulation support
Handles the range dialogue for interesting quantities in accumulations. Interesting quantities are those which go on the axes of the output data file (e.g. X and Y for an image, energy for a spectrum, time for a time profile or potentially any quantity for a photon file).

`NDIMENS` must be set to 1 for 1-d output data files (histograms or time profiles), 2 for 2-d images and 0 for photon files.
`NFORMAT` is a reserved value, which must be set to 0 for photon files or 3 for time profiles (with the exception of the latter, use the same value of `NDIMENS` for convenience).
`XSTUFF` is an array of the names of the `NDIMENS` quantities considered as interesting (i.e. whatever goes on the X axis of an histogram or on the XY axes of an image)
`STUFF` is an array of the names of all quantities and is used mainly for photon files. On **return** `XSTUFF` will be modified to contain the quantities accepted as interesting.
In all cases see also sax_acc_preload and sax_acc_select for the `(X)STUFF` arrays.

Essentially this routine constructs adequate defaults for the start and end value and the binning of each quantity, and asks the user to confirm or change them (directly for all parameters except times, which are processed through asktime and askbin). In the case the accumulation is that of a photon file, instead of the binning the user is asked whether the quantity has to be included in the output or not.
The ranges may be used both for data acceptance (values out of range cause data rejection) and to dimension the size of the output data file.

- ## sax_acc_select

Library	fotlib	Fortran code
Calling sequence		
Arguments	`INTEGER`	`NDIMENS`
	`INTEGER`	`NFORMAT`
	`CHARACTER*(*)`	`XSTUFF(*)`
	`CHARACTER*(*)`	`STUFF(*)`
	`INTEGER`	`IERR`

Category: accumulation support
This routine selects (among the quantities listed in `STUFF` and preset by sax_acc_preload) the `NDIMENS` interesting ones, asking the user (except for time profiles, where the 'TIME' quantity is used and for photon files where this routine is not called). Their names is **returned** in `XSTUFF`.
The arguments are those listed for sax_acc_range, plus `IERR` which **returns** a non-zero value in case of errors.

- ## sax_df_keywords

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_DF_KEYWORDS`	

This routine writes into the header of the XAS file the standard keywords used for SAX specific accumulations, i.e. the satellite and instrument identification, and the observer and target identification (taken from tape directory). The observer name can be overridden (e.g. by the current user full name or another name) if so requested in the XAS environment.

- ## sax_open_dir

Library	fotlib	Fortran code
Calling sequence	`CALL AX_OPEN_DIR(LU,NOBS,NAME)`	
Arguments	`INTEGER`	`LU`
	`INTEGER`	`NOBS`
	`CHARACTER*(*)`	`NAME`

This routine opens the observation directory file for observation `NOBS` on the logical unit `LU` (automatically selected), **returning** also its file `NAME`

- ## sax_pcf_load

Library	fotlib	Fortran code
Calling sequence	`CALL SAX_PCF_LOAD(HKNAME)`	

Arguments	CHARACTER*(*)	HKNAME

This routine loads in memory the PCF entry for HK parameter HKNAME. The PCF (Parameter Characteristics File) is an instrument support file listing the characteristics of all HK parameters. The routine selects the current instrument PCF, or, when appropriate, the spacecraft PCF.

- ## sax_pcf_lookup

Library	fotlib	Fortran code
Calling sequence	CALL SAX_PCF_LOOKUP(FIELD,TYPE,IVAL,STRVAL,FOUND)	
Arguments	CHARACTER*(*)	FIELD
	INTEGER	TYPE
	INTEGER	IVAL
	CHARACTER*(*)	STRVAL
	LOGICAL	FOUND

Similar to pktcap_lookup, but looks up the characteristics named FIELD of the HK parameter whose PCF entry was loaded in memory by sax_pcf_load

---

- ## sax_pktcap_load

Library	fotlib	Fortran code
Calling sequence	CALL SAX_PKTCAP_LOAD(PACKET)	
Arguments	CHARACTER*(*)	PACKET

This routine loads in memory the packetcap entry for the named telemetry PACKET. The routine builds the appropriate packetcap file name and locates it, then uses pktcap_load to do the actual job.

- ## sax_which_data

Library	fotlib	Fortran code
Calling sequence	CALL SAX_WHICH_DATA(PACKET,DATATYPE,N)	
Arguments	CHARACTER*(*)	PACKET
	CHARACTER	DATATYPE ('I'|'S'|'P'|'T'|'HK')
	INTEGER	N

As in check_packet, it **returns** the number N of packet families listed in the instrument directory and eligible for the requested accumulation DATATYPE (where the codes are for Images, Spectra, Photon files, Time profiles or HK time profiles). In addition it also returns the PACKET name selected for accumulation (this is automatic if N=1, otherwise the user is prompted for selection among a list), or a blank string if no packets available (N=0)

- ## tapechar

Library	fotlib	Fortran code
Calling sequence	CALL TAPECHAR	

Category: specific of fotfile program
Loads the system (and site) dependent template tape commands (to be later used by tape command generation routines) from a programming support file located in the local directory.

- ## timebin_b1s1

Library	fotlib	Fortran code
Calling sequence	CALL TIMEBIN_B1S1(COVERAGE, START, END, SIZE)	
Arguments	DOUBLE PRECISION	COVERAGE
	DOUBLE PRECISION	START, END
	DOUBLE PRECISION	SIZE

Computes the live COVERAGE (exposure time) of a time bin of given STARTD and ENDtimes and SIZE. Intended to be called by time profile INCREMENT_ROUTINEs called by sax_acc_b1s1_y (which fills the relevant COMMON block with appropriate information).

- ## unlintomm

Library	fotlib	Fortran code
Calling sequence	CALL UNLINTOMM(X,Y)	

Arguments	DOUBLE PRECISION	X,Y

Category: attitude
Converts **in place** the coordinates X,Y from unlinearized pixels to mm on the focal plane. Although the routine is potentially general, it is implemented only for MECS (and contains MECS specific imbedded code) since it needs to load the linearization coefficients and perform the linearization.

[Previous][Next] [Up][Down]

5.7 The MECS library

The MECS (mecslib) library groups routines specific of the SAX MECS instrument (calibration data access or event corrections).

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

A-B	abs_co_be	alum	area_mr	bewin_trasp	blsel
C-E	coda	cross_sec	ein	eout	escape
F-K	fopen_rmf	gas_cell	init_correct_me	init_correct_me_fast	kapton2
L-M	lexan	me_gain_time	me_init_map	mecs_arf	mecs_keywords
	mecs_matkeywords	mecs_rmf	mecscorrect		
P-S	poly	poly_carbo	psf_mir	psf_rad	spread
T-Z	tetafi_xy	write_arf	write_rmf	write_rmf_ebo	write_rmf_mat

---

- ## abs_co_be

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- ## alum

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- ## area_mr

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- ## bewin_trasp

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- ## blsel

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- ## coda

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **cross_sec**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **ein**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **eout**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **escape**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **fopen_rmf**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **gas_cell**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **init_correct_me**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **init_correct_me_fast**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **kapton2**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **lexan**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **me_gain_time**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **me_init_map**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **mecs_arf**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **mecs_keywords**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **mecs_matkeywords**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **mecs_rmf**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **mecscorrect**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **poly**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **poly_carbo**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **psf_mir**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **psf_rad**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **spread**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **tetafi_xy**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **write_arf**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **write_rmf**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **write_rmf_ebo**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

- **write_rmf_mat**

Library	mecslib	Fortran code
Calling sequence		
Arguments		

[Previous][Next] [Up][Down]

5.8 The LECS library

The LECS (`lecslib`) library groups routines specific of the SAX LECS instrument (calibration data access or event corrections). These routines are not officially supported.

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

blrng	e2fwhm	init_correct_le	le_gain_time	lecs_keywords	lecscorrect

- **blrng**

Library	lecslib	Fortran code
Calling sequence		
Arguments		

- **e2fwhm**

Library	lecslib	Fortran code
Calling sequence		
Arguments		

- **init_correct_le**

Library	lecslib	Fortran code
Calling sequence		
Arguments		

- **le_gain_time**

Library	lecslib	Fortran code
Calling sequence		
Arguments		

- **lecs_keywords**

Library	lecslib	Fortran code
Calling sequence		
Arguments		

- **lecscorrect**

Library	lecslib	Fortran code
Calling sequence		
Arguments		

[Previous][Next] [Up][Down]

5.9 The PDS library

The PDS (pdslib) library groups routines specific of the SAX PDS instrument (calibration data access, event corrections, instrument specific keywords). These routines have been written at ITESRE and are not supported by the author of this document (refer to comments in the code).

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

- ## init_correct_pds

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## inst_key_copy

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## inst_key_find

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## inst_key_flush

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## inst_key_load

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## inst_key_mult

Library	pdslib	Fortran code
Calling sequence		

Arguments		

- **inst_key_read**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **inst_key_set**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **instrument_keys**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **no_keyword**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pds_arf**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pds_ein**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pds_en_resol**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pds_fotunits**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pds_freq**

Library	pdslib	Fortran code
Calling sequence		

Arguments		

- ## pds_keywords

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_matinfo

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_matkeywords

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_matout

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_ogip

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_opnrmf

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_response

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_wrtrmf

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- ## pds_wrtrmfebo

Library	pdslib	Fortran code
Calling sequence		

Arguments	

- **pds_wrtrmfmat**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pdscorrect**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pdsmat_coef**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **pdsmat_init**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **rminmax**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **sax_froot_name**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **time_70s2mjd**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **time_a2mjd**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **time_cldj**

Library	pdslib	Fortran code
Calling sequence		

Arguments	

- **x_echo**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **x_echo_error**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

- **xdofit**

Library	pdslib	Fortran code
Calling sequence		
Arguments		

[Previous][Next] [Up][Down]

5.10 The HPGSPC library

The HPGSPC (`hpgslib`) library groups routines specific of the SAX HPGSPC instrument (calibration data access or event corrections). Part of these routines have been written at IFCAI and are not supported by the author of this document (refer to comments in the code). The remainded of the routines are not officially supported.

Use the subject list in the previous page, or the quick alphabetic index here below to locate the routine of interest.

- **broad2**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **buf_read**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **cofas**

Library	hpgslib	Fortran code
Calling sequence		

Arguments		

- **config_read**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **effmed**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **filecorr_read**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **fileinp_read**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **fopen_rmf**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **fuga**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **fuga_1**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **hp_gain_time**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **hp_keywords**

Library	hpgslib	Fortran code
Calling sequence		

Arguments		

- **hpcorrect**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **init_correct_hp**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **julia**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **parameter**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **reader**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **reader_1**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **shell_fact**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **shell_prob**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **tmed**

Library	hpgslib	Fortran code
Calling sequence		

Arguments		

- **tofits**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **toqdp**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **winbe**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **write_arf**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **write_rmf_ebo**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

- **write_rmf_mat**

Library	hpgslib	Fortran code
Calling sequence		
Arguments		

[Previous][Next] [Up][Down]

6. the include files

This section gives access (by alphabetic order and by subject) to the description of the Fortran `INCLUDE` files used by XAS. They are usually definitions of `COMMON` blocks or standard `PARAMETER` constants.
The XAS convention is that these files have a `.inc` extension.

File alphabetic list

A-C	accumcommon	accumindir	auxrecord	b1s1local	bigbuf	bincommon	clientside	context
D-I	debcommon	errors	graphlimit	hcommon	hkcommon	hsizes	instkeys	
L-O	labc1	labco	lecommon	mecommon	megaincommon	opcommon		
P-R	pdscoef	pdsmat	pdsnaipsa	pencommon	pixcommon	psserver	radiant	runstring
S-Z	saxfot	servers	syscommon	timecommon	vcommon	voscommon	wcommon	xwserver

File subject list

You can locate a file by subject using the following lookup table

VOS, user interface and general	XAS file	accumulation	attitude	graphics	instruments

VOS, user interface and general

VOS error codes	errors voscommon
trigonometric constants	radiant
user interface	runstring
data conversion	syscommon

XAS file support

image handling buffers	bigbuf
binary tables	bincommon
general context	context
XAS file headers	hcommon hsizes

accumulation support

telemetry file access	accumcommon accumindir auxrecord b1s1local
HK parameters	hkcommon
times and time windows	opcommon timecommon vcommon wcommon

attitude

debcommon pixcommon

graphics

client side	clientside labc1 labco pencommon servers
server side	graphlimit psserver servers xwserver

SAX instrument support

LECS & MECS	lecommon	mecommon megaincommon
PDS	instkeys pdscoef pdsmat pdsnaipsa	
FOT tapes	saxfot	

- ## accumcommon

include file		Fortran code
Usage	define ACCUMCOMMON block used by accumulation program and relevant PARAMETER constants	
Content	COMMON	ACCUMCOMMON
	COMMON	ACCUMCOMMONC
	INTEGER PARAMETER	ACCUMCOMMON_MAXDIM

According to usual practice there are two separate common blocks, one for numeric and one for character variables. Both sets of variables have names prefixed with `ACCUMCOMMON_`. The meaning and full list of variables is documented in the code header. These variables are filled dynamically by the various programs, and contain information about the current accumulation packet (typically a direct mode one), the quantities chosen, the time limits, the decoded values of quantities for the current event.
Some of these quantities are arrays dimensioned to `ACCUMCOMMON_MAXDIM`, the maximum number of fields present for an event (currently fixed to 16, which is the maximum for HPGSPC packets).

- ## accumindir

include file		Fortran code
Usage	define ACCUMINDIR block used by accumulation program and relevant PARAMETER constants	

Content	COMMON	ACCUMINDIR
	INTEGER PARAMETER	ACCUMINDIR_MAXDIM

This is an additional common block used to hold the data structures necessary for accumulation involving indirect mode data which do not fit into ACCUMCOMMON. Of these the most important is a buffer array ACCUMINDIR_HISTOGRAM, used to hold a raw spectrum read from a packet, and dimensioned to the maximum size ACCUMINDIR_MAXDIM (currently fixed to 4096 channels as required by HPGSPC).

- **auxrecord**

include file		Fortran code
Usage	buffer space for "ASCII HK" accumulations	
Content	COMMON	AUXRECORD

Contains two arrays of integer and real values decoded from an ASCII HK telemetry record (actually pseudo-telemetry, i/.e. ground-generated orbit or attitude data).

- **b1s1local**

include file		Fortran code
Usage	cyclic buffer shared between time binning and exposure processing routines and relevant PARAMETER constants	
Content	COMMON	B1S1LOCAL
	INTEGER PARAMETER	ICIRC

This common block is used to communicate between the exposure_b1s1 and exposure_b1s3 routines on one side and timebin_b1s1 on the other side. It mantains a circular buffer of start and end times of the last ICIRC packets, with also an array of flags indicating whether the packets follow without jumps or not, and a pointer to the current packet.
The size ICIRC has been empirically tuned to 3000 packets.

- **bigbuf**

include file		Fortran code
Usage	shared buffer for image file i/o	
Content	COMMON	BIGBUF

This common is used by create_image and read_image to share space for a silly data buffer.

- **bincommon**

include file		Fortran code
Usage	define BINCOMMON common block used by all programs dealing with binary table XAS files and relevant PARAMETER constants	
Content	COMMON	BINCOMMON
	INTEGER PARAMETER	BINCOMMON_MAXTABLES, BINCOMMON_MAXFIELDS

The BINCOMMON common block holds the binary table descriptors managed by many routines like copy_table_desc get_table_desc set_table_desc etc.

There are as many elements in the descriptor arrays as the maximum number of tables BINCOMMON_MAXTABLES (which is currently **equated** to the maximum number of XAS files defined in HCOMMON, i.e. up to all XAS files can be tabular), and as the maximum number of columns in a table BINCOMMON_MAXFIELDS (currently fixed to 16 for consistency with the maximum number of event fields for a photon defined in ACCUMCOMMON.
For this reason 'hcommon.inc' **must be explicitly included before** referencing this include file.

The descriptor arrays are the type or bit width (BITPIX in FITS parliance) of a column, its dimensionality or depth, and a number associating a logical column with its physical position or order.

- **clientside**

include file		Fortran code
Usage	define data buffer for communication between graphics clients and server	
Content	COMMON	CLIENTSIDE

All the low level graphics routines in graphserv library share this buffer to save space. The buffer holds space for the data sent by a client on the communication channels, which according to the specification of XAS graphics are represented by an opcode followed by a number of operands. These operands may be integer, real or character and all share the same space in the buffer after the opcode.

The dimension of the buffer is set by PARAMETER constants defined in ['graphlimit.inc'] which is **automatically referenced** by `clientside.inc`.

- ## context

include file		Fortran code
Usage	define the "context" CTXCOMMON common block	
Content	`COMMON`	`CTXCOMMON`

A trivial COMMON used by many routines in xaslib to hold the current instrument name and the current context (the context is the type of data requested for output, e.g. spectra, images, matrices etc.) once they have been retrieved from XAS environment variables.

- ## debcommon

include file		Fortran code
Usage	dedicated to planned deblurring correction	
Content	`COMMON`	`DEBCOMMON`
	`COMMON`	`DEBCCOMMON`

Reserved for future accumulation programs capable of deblurring image positions according to attitude.

- ## errors

include file		Fortran code
Usage	definition of VOS error codes	
Content	`INTEGER PARAMETER`	`VE_`*`NAME`*

All symbolic constants defining VOS error codes are associated a numeric value in this file. For better legibility programs shall use the named mnemonics (i.e. `VE_NOFILE` for a "file not found" error is better than "code 102")
See the error listings for details.

- ## graphlimit

include file		Fortran code
Usage	define PARAMETER constant to size data buffer for communication between graphics clients and server	
Content	`INTEGER PARAMETER`	`FULLBUFSIZE` and derived

The buffer used to exchange data over communication channels between graphics servers (which **must include** this file together with `'psserver,inc'` or `'xserver.inc'`) and clients (which include `'clientside.inc'`) is sized here to 1026 bytes of full length (additional constants are derived internally to define the same length or parth thereof for other datatypes).
Note that the f2x C routine defines the same common blocks used by xserver, as a C `struct` which is sized by separate `#define`s which must be edited manually consistently with eventual changes in `FULLBUFSIZE`.

- ## hcommon

include file		Fortran code
Usage	define HCOMMON common block used by all programs dealing with XAS file headers, and relevant PARAMETER constants.	
Content	`COMMON`	`HCOMMON`
	`COMMON`	`HCCOMMON`
	`COMMON`	`MINIH`
	`INTEGER PARAMETER`	`HCOMMON_TOP`
	`INTEGER PARAMETER`	`HCOMMON_MAXFILES`

This file defines separately two logical areas, the HCOMMON buffer array (with space for the headers of all open XAS files) and the MINIH buffer holding the mini-header of the "current" file.

According to usual practice, in HCOMMON there are two separate common blocks, one for numeric and one for character variables. Both sets of variables have names prefixed with `HCOMMON_`. The meaning and full list of variables is documented in the code header.
The variables which must be arrays with one element per open XAS file (and not just refer to the current file or to global properties) are size to `HCOMMON_MAXFILES` (the current maximum is 8 XAS files).
The character part includes an array of header buffers (each one originally sized to `HCOMMON_TOP` bytes (currently 2048) and an array of mini-header buffers (the "current" of which is copied to MINIH described below). Header buffers contain a full binary image of a file hader read from disk, and if a space

larger than HCOMMON_TOP is necessary, routines like h_load_header take care of dynamic re-allocation.

The MINIH mini-header is a trivial 28-byte object whose pieces are defined here by EQUIVALENCE.

- ## hkcommon

include file		Fortran code
Usage	define common block for HouseKeeping data handling	
Content	COMMON	HKCOMMON

The HKCOMMON block contains variables (whose meaning and full list of variables is documented in the code header) used to handle HK (HouseKeeping) parameters, and to decode them according to the specification of the PCFs.

- ## hsizes

include file		Fortran code
Usage	define PARAMETER constant for maximum header keyword size	
Content	INTEGER PARAMETER	MAXKEYBUF and derived values

Routines dealing with header keywords (and programs doing special dealings with keywords) require to know the maximum size of the keyword data area. Since a complete keyword requires the binary space for 8 bytes for the name, 1 byte for the type and 1 byte for the length, and cannot exceed 256 bytes (because the length itself is coded in a byte), the maximum data area could potentially be 256-10 byte long. However the choice is to define MAXKEYBUF to 78 bytes, which gives a a data area of 68 bytes which is consistent with the longest FITS character keyword value. The restrictions on the number of elements in numeric array keywords descend from such choice.

- ## instkeys

include file		Fortran code
Usage	support to PDS instrument-specific header keywords	

unsupported by the author of the present document

- ## labc1

include file		Fortran code
Usage	used by lb_ routines	
Content	COMMON	AXES (no detail given)

- ## labco

include file		Fortran code
Usage	used by lb_ routines	
Content	COMMON	GENERAL (no detail given)

A couple of vestigial common blocks as used by the original "Labeller" program from which these routines were derived.

- ## lecommon

include file		Fortran code
Usage	define common block for LECS event corrections and relevant PARAMETER constants	
Content	COMMON	LECOMMON

This common block contains all coefficients necessary for the various LECS event corrections. LECS event correction is not officially supported and is freely mimicked on the MECS event correction. The meaning and full list of variables is documented in the code header.

- ## mecommon

include file		Fortran code

	COMMON	MECCOMMON

This common block contains all coefficients necessary for the various MECS event corrections. The meaning and full list of variables is documented in the code header. According to usual practice there are two separate common blocks, one for numeric and one for character variables.

- ## megaincommon

include file		Fortran code
Usage	common block for the MECS gain relation	
Content	COMMON	GAINCOMMON

Contains coefficients for the gain (energy-PHA channel) relation used during MECS matrix computation by routines like coda, spread and eout. The meaning and full list of variables is documented in the code header.

- ## opcommon

include file		Fortran code
Usage	common block for observing period times	
Content	COMMON	OPCOMMON

A simple common block used by accumulation program to keep the start and end time of the current observation chain in a variety of formats. The meaning and full list of variables is documented in the code header.

- ## pdscoef

The following include files, used for PDS matrix computation, are unsupported by the author of the present document

include file	Fortran code
Usage	
Content	

- ## pdsmat

include file	Fortran code
Usage	
Content	

- ## pdsnaipsa

include file	Fortran code
Usage	
Content	

- ## pencommon

include file		Fortran code
Usage	define PENCOMMON common block for graphics clients	
Content	COMMON	PENCOMMON

Keeps basic information like pen colours and plotting window corners. The meaning and full list of variables is documented in the code header.

- ## pixcommon

include file		Fortran code
Usage	common block for pixel to celestial coordinate transformation	
Content	COMMON	PIXCOMMON
	COMMON	PIXCCOMMON

This area contains coefficients used by unofficial programs used to handle celestial coordinates According to usual practice, there are two separate common blocks, one for numeric and one for character variables. The meaning and full list of variables is documented in the code header.

- ## psserver

include file		Fortran code
Usage	define F2C common block and PARAMETER constants relevant to Postscript graphics server	
Content	COMMON	F2C
	COMMON	F2CC
	REAL PARAMETER	PTS_PER_INCH, CM_PER_INCH and derived

The logical F2C common block is used by a graphics server to keep its internal status. The PostScript server uses for convenience the same names for the F2C common defined for `'xwserver.inc'` although the data types of some elements may be different. According to usual practice, in F2C there are two separate common blocks, one for numeric and one for character variables. The meaning and full list of variables is documented in the code header.
The `'graphlimit.inc'` file **must be referenced before** this one, since some PARAMETERs defined there are used for dimensioning.

In addition some trivial parameters useful in writing PostScript code are defined here.

- ## radiant

include file		Fortran code
Usage	defines PI and degree-to-radians constants	
Content	PARAMETER DOUBLE PRECISION	PI,DTOR

Defines two trivial constants. Use the idiom `RADIANT = DEGREE*DTOR` to convert degrees to radians.

- ## runstring

include file		Fortran code
Usage	define XRCOMMON common block used by user interface routines and relevant PARAMETER constants	
Content	COMMON	XRCOMMON
	COMMON	XRCCOMMON
	INTEGER PARAMETER	MAXRUN

This common block is used by the x_read routine and related to hold variables related to the user interface (initialized in system dependent way by blkxrcommon).
According to usual practice, in XRCOMMON there are two separate common blocks, one for numeric and one for character variables. Both sets of variables have names prefixed with `XRCOMMON_`. The meaning and full list of variables is documented in the code header.

In addition parameter `MAXRUN` defines the maximum size of a run string. This is used also by some other routines and programs.

- ## saxfot

include file		Fortran code
Usage	common block for `fotfile` program	
Content	COMMON	SAXFOT

This common block is used by tape command generation routines to store internal information. The meaning and full list of variables is documented in the code header.

- ## servers

include file		Fortran code
Usage	define PIPECOMMON common block with graphics server communication channel status	
Content	COMMON	PIPECOMMON
	COMMON	PIPECCOMMON
	INTEGER PARAMETER	MAXSERVER

This common block is used by the programs which control graphics servers and by the communication channel handling routines to keep track of the status of registered graphics servers.
According to usual practice, in PIPECOMMON there are two separate common blocks, one for numeric and one for character variables. Both sets of variables have names prefixed with `PIPECOMMON_`. The

meaning and full list of variables is documented in the code header.
Some of the variables are arrays sized to MAXSERVER (currently 9), the maximum number of servers active at any time.

- ## syscommon

include file		Fortran code
Usage	define SYSCOMMON common block used for data conversion	
Content	COMMON	SYSCOMMON
	COMMON	SYSCCOMMON
	INTEGER PARAMETER	SYSCOMMON_NSYS

This common block is used by type conversion routines and programs to know whether a (and which type of) conversion is required between an origin operating system and the current target system.
According to usual practice, in SYSCOMMON there are two separate common blocks, one for numeric and one for character variables. Both sets of variables have names prefixed with SYSCOMMON_. The meaning and full list of variables is documented in the code header.
The parameter SYSCOMMON_NSYS is the number of currently supported operating systems and **must be edited** in cases of porting to a OS. In addition one **shall change** the blksyscommon routine used to initialize the different OS characteristics.

- ## timecommon

include file		Fortran code
Usage	define TIMECOMMON common block used for time unit conversion	
Content	COMMON	TIMECOMMON
	COMMON	TIMECCOMMON

This common block is used to store constants used for conversion of times between spacecraft times and user defined units (seconds or submultiples) and is initialized by time_constants_setup
According to usual practice, in TIMECOMMON there are two separate common blocks, one for numeric and one for character variables. Both sets of variables have names prefixed with TIMECOMMON_. The meaning and full list of variables is documented in the code header.

- ## vcommon

include file		Fortran code
Usage	define VCOMMON common block used to support to saxvartaccum program	
Content	COMMON	VCOMMON

saxvartaccum is an unofficial XAS extension which allows to accumulate time profiles with bins of variable width defined in an external file. The VCOMMON blocks keeps track of such bin characteristics similar to (but separately from) what is done for time windows in wcommon. The meaning and full list of variables is documented in the code header.

- ## voscommon

include file		Fortran code
Usage	common containing current VOS errors	
Content	COMMON	VOSCOMMON

This simple common block contains two variables VOSCOMMON_ERROR and VOSCOMMON_SYSTEMERROR which are set by VOS and other routines. The first variable is set to the standard VOS codes defined in the 'errors.inc' file, while the second contains the original system dependent code. Users do not normally access this common explicitly, but via a call to voserror

---

- ## wcommon

include file		Fortran code
Usage	define WCOMMON common block used to store time windows	
Content	COMMON	WCOMMON

The WCOMMON block stores information about the time windows used to select intervals from data accumulation as read from a file via appropriate routines.
Variables names are prefixed with WCOMMON_. The meaning and full list of variables is documented in the code header.

- ## xwserver

include file		Fortran code

Usage	define F2C common block as used by X window graphics server	
Content	COMMON	F2C
	COMMON	F2CC

The logical F2C common block is used by a graphics server to keep its internal status. The X window server uses for convenience the same names for the F2C common defined for `'psserver.inc'` although the data types of some elements may be different. More important, it has to be noted that the f2x C routine defines F2C and F2CC as C `struct`s sized by separate `#define`s which must be edited manually consistently with eventual changes in `FULLBUFSIZE` in `'graphlimit.inc'`
For such reason the `'graphlimit.inc'` file **must be referenced before** this one,.

According to usual practice, in F2C there are two separate common blocks, one for numeric and one for character variables. The meaning and full list of variables is documented in the code header.

[Previous][Next] [Up][Down]

---

7. the programming support files

This section describes miscellaneous support files (located in the `include` or `local` directories).

- error code listings
- font listings
- graphics marker listings
- postscript prologues
- tape command definition

[Previous][Next] [Up][Down]

7.1 error code listings

There are two auxiliary files in the `include` directory which are not used by any program, but list the VOS error codes defined in errors.inc in human readable form :

- errors.list lists the numeric error code (as printed by voserror calls) with an explanatory message.
- errors.list_long in addition to the same information lists also the symbolic code used in errors.inc and the routine where the error may occur

---

[Previous][Next] [Up][Down]

7.2 font listings

The font listings xwfont.list and psfont.list are used respectively by the X window and postscript graphics servers.
These files are intended as prototypes for customisation and can be freely copied, since they are looked for in the following order :

- in the current working directory where the user is running XAS (any user can copy it here as a special dedicated setup)
- in the user home directory (any user can copy it here as a personal private all-purpose setup)
- in the `local` directory (the XAS installer can copy it here as a systemwide site-dependent customised setup)
- in the `include` directory (this is the fallback version delivered with XAS)

Both files associate a font number (the only thing known to the graphics server) with a real font. They do it slightly differently :

- for the X window server a real font is specified by the X (xlsfonts ?) descriptive string (inclusive of font face and size)
- for the Postscript server one specifies separately a size in points and a font face name

---

[Previous][Next] [Up][Down]

7.3 graphics marker listings

The single psmarker.list is intended as a way to define (and customize the definition) for markers for the graphics polymarker primitive. The code (which should apply only to Postscript graphics servers is unimplemented/

The file shall define Postcript macros named `Mnnn` defining a 1x1 point (scalable) marker, according to the prescription given in the example.

[Previous][Next] [Up][Down]

7.4 postscript prologues

The prologue files located in the `include` directory are used by the Postscript graphics servers to define the standard macros used to realize the graphics primitives.
There are three different prologue files :

- bwserver.prologue
  The standard black-and-white prologue used by `bwserver`
  Define a standard 7-pen lookup table with selected shades of gray, and a 256-level gray lookup table
- cpserver.prologue
  The simple colour prologue used by `cpserver`
  Define a standard 7-pen lookup table with selected fundamental colours, and a 256-level colour RGB lookup table initialized to shades of red
- c2server.prologue
  The Level-2 colour prologue used by `c2server`
  is similar to the previous but uses a Level-2 operator for images (with /ASCIIHexDecode filter) which allows image data to point directly into the colour lookup table (this means the data is 3 times less bulky)

At present they all define inside a simplified version of the marker definition file

[Previous][Next] [Up][Down]

7.5 tape command definition

This file is a natural candidate for site-dependent customisation and is therefore located in the `local` directory. This file is used to support tape reading in `fotfile`, which operates reading template commands from this file, and using tape command generation routines to write the relevant edited commands to a shell script. Example files are shown here for

- Unix using a local tape drive and `dd`
- Unix using a remote tape drive via rsh (this is a non functional example, and is also not recommended w.r.t. remote login to the machine with a local tape)
- VMS which requires a dedicated tapecopy (dd emulator) program which is not part of XAS.

All examples are self-documented. In a nutshell one shall equate some standard symbolic functions to a system-dependent command, using tokens in place of variable parts like tape names, file names or numeric parameters. The symbolic functions are :

- REWIND for tape rewind
- SKIP for forward skip file
- BINCOPY for copying and unblocking of a binary file
- ASCCOPY for copying and unblocking of an ASCII file
- MESSAGE to write a message to standard output

[Previous][Next] [Up][Down]

8. the calibration files

This section describes the calibration files proper (located in the `calib` subdirectories). The term refers to files describing physical characteristics of specific SAX instruments.
Files describing the format of the data associated to the SAX instruments are described elsewhere, although they may be stored in the same directory.

The underlying idea about calibration files is that all files which comprise only one or a few numeric values are kept in ASCII files (usually with a name like `something.coeff`) while bulkier data (like relative gain images) are kept with XAS naming conventions in XAS format data files (these will be by definition system dependent, since XAS files use native binary format).

Some information follows about SAX calibration files.

- spacecraft
  no officially supported file besides the instrument support files; the ITESRE barycentrization program keeps an `earth.dat` file here
- MECS used by mecslib routines
  calibration files are described in detail in the MECS Guided Tour and can be accessed here
- LECS used by lecslib routine
  calibration files are not officially supported
- HP-GSPC used by hpgslib routine
  calibration files supplied by IFCAI are not supported by the author of this document
- PDS used by pdslib routines
  calibration files supplied by ITESRE are not supported by the author of this document

---

[Previous][Next] [Up][Down]

---

9. the instrument support files

This section describes files located in the `calib` subdirectories which describe the format of the data associated to the SAX instruments. Files describing physical characteristics of specific SAX instruments are described in the previous section.

- telemetry packet description (packetcap)
  - spacecraft
  - MECS
  - HPGSPC
  - PDS
- HK parameter characteristics files (PCF)
  - spacecraft
  - MECS data under spacecraft telemetry
  - MECS
  - HPGSPC
  - PDS
- Experiment Configuration parameter files

[Previous][Next] [Up][Down]

9.1 packetcap

The packetcap files are telemetry packet description files mimicked on the Unix termcap files.
Full definition of packetcap files is kept in a separate paper document. In brief however one shall note the following guidelines :

- comment lines begin with #
- a packetcap entry relative to single packet type may span more lines
- the first line shall begin with the (packet) identifier followed by a pipe (|) and a comment, followed by field entries
- all lines but the last must be terminated by a continuation mark, i.e. a backslash (\) **not** followed by any blank ! Continuation lines are preferably aligned not in column one.
- field entries are delimited by semicolons (:)
- field names are two or three characters (this is an extension to the Unix termcap)
- There are three types of fields
  - boolean fields with no values, e.g. `:ve:`
  - integer fields with numeric values, e.g. `:pl#1912:`
  - string fields with values, e.g. `:f1=TIME:`
- a particular field is `:tc=`*`identifier`*`:`, which chains the current definition to another packetcap entry (this is useful when more packets share a common set of definitions)
- current routines use a basic type `:bt#`*`n`*`:` and secondary type `:st#`*`m`*`:` field to classify similar packet layouts together
- other fields like packet length `:pl#`*`n`*`:` and number of items `:ni#`*`m`*`:` are relevent to an entire packet
- a packet item can be an event, or a spectrum, or an HK sample, and then contain `:nf#`*`n`*`:` fields, each of them (i=1,n) has characteristics like a field name `:f`*`i`*`=`*`name`*`:`, a field size `:s`*`i`*`#`*`k`*`:`, etc.

Packetcap files are stored in an instrument specific subdirectory of the `calib` directory, therefore can be mantained separately for different missions and instruments.

The current packetcap files for SAX instruments can be inspected here :

- spacecraft HK
- MECS direct mode, indirect mode and HK
- HPGSPC direct mode, HK
- PDS direct mode, indirect mode and HK

[Previous][Next] [Up][Down]

9.2 PCF

The PCF (Parameter Characteristics Files, an old ESA name) are HouseKeeping parameter description files mimicked on packetcaps, in turn the Unix termcap files.
Full definition of PCF files is kept WHERE???. In brief however one shall note the following guidelines :

- the general syntax (comment lines, continuation lines, fields etc.) is the same of packetcap files
- the first line of a PCF entry begins with the HK parameter mnemonic followed by a pipe (|) and a comment, followed by field entries
- a compulsory field is `:pk=`*`packet`*`:` which points to a packetcap entry for the telemetry packet which contains the particular HK parameter.
- other fields consider the parameter commutation, its position in the record, its physical units and the calibration curve necessary to convert it to physical units. For explanation look in comments in actual PCFs

---

The current PCF files for SAX instruments can be inspected here :

- spacecraft HK
- MECS HK data under spacecraft telemetry (this is actually a link to the next file but must exist in the spacecraft subdirectory)
- MECS HK
- HPGSPC HK
- PDS HK

[Previous][Next] [Up][Down]

9.3 Experiment Configuration Parameter file

These files are used exclusively by the check_expconf programs. They have a name like `instrument.expconf` and are located in the `$XASTOP/calib/sax/`*`instrument`* directory.

Format

A single ASCII file per instrument with the following structure

- Two header lines (copy them from an existing file)
- as many records as parameters in the FOT Experiment Configuration files.

  Each record has the following layout, where fields are separated by one or more blanks. Alignment into columns is encouraged for legibility.

  - 1-character code to be selected among `C`, `I` or `R` to indicate the parameter is character, integer, real.
  - 8-character UPPERCASE name of the parameter, as it appears in the FOT Experiment Configuration files. Parameters shall appear in this parameter file in the same order as in the FOT files.
  - parameter classification, which indicates the action by check_expconf, to be chosen among one of the following values (verbatim !) :

    `relevant`
    an important parameter used in routine operations which shall appear in the summary file, and whose value shall be verified to be nominal or not. E.g. HV related parameters, mode and POP status.

    `checked`
    intended for parameters which are normally left alone in their nominal state, but for which a check of being nominal is useful. Currently handled as "relevant" ones. E.g. the LED status (not used by GOs, but might be commanded).

`dontcare`
intended for parameters which are operated seldom. Should not appear in summary, unless thery are not nominal. Currently handled as "ignored" ones. E.g. analog and digital thresholds, unused NFI HV relays.

`ignored`
parameters which are likely to be never operated, they do not appear in output summaries and their values are skipped by the processing. E.g. all parameter of indirect modes for MECS.

- nominal value(s) of the parameter. The values shall be in the same form appearing in the FOT files. A list of values may be declared nominal, in which case values are separated by a pipe character. E.g. `DIR3|DIR2`. For ignored parameters, use the string `any`.
- conditioning parameter. Either the lowercase word `no` or an 8-character UPPERCASE name of another parameter, which is conditioning the validity of the current one.
- conditioning parameter values. Blank if parameter is not conditioned. Otherwise a pipe-separated list of values of the conditioning parameters, for which the current parameter value is considered "meaningless" (therefore its value will not be displayed, but considered "invalid")

As examples, consider that HV values are conditioned by the respective HV switch (they are invalid when HVs are OFF), or that POPID or ITOPMODE are meaningless if POPSTAT is DISABLE. Use the `mecs.expconf` file as reference.

[Previous][Next] [Up][Down]

---

10. details about specific programs

This section is intended to give an overview of some specific XAS programs, while overview of general idioms and graphics programs are covered elsewhere.

- the `fotfile` FOT tape reading program (SAX specific)
- the `check_expconf` experiment configuration summary generator (SAX specific)
- xasset and xasplot ?
- typical accumulate dispatcher
- typical plot dispatcher
- typical SAX telemetry accumulator
- typical SAX matrix accumulator
- saxauxcalc
- time windows
- keyword header access hlist, tlist, header_edit
- localize

[Previous][Next] [Up][Down]

---

11. XAS graphics

XAS graphics is based on a client-server approach.

- The typical commands which the user accesses are clients
- These clients are linked with high level graphics routines (the `xasgraph` library)
- These routines in turn use low level graphics routines (the `graphserv` library) to send an opcode and eventual operands to a graphics server over a pair of communication channels : one for sending command, and one for receiving server replies.
- Up to this level everything is device independent
- The various server instances contain into themselves all device specific stuff.

In practice only one X-Window based server (for interactive work) and a family of Postscript based servers (for hardcopies) are implemented.
The communication routines used by (clients and) servers are also in the `graphserv` library, together with the Xlib C interface used by the X-Window based server.
The Postscript based servers instead use self-contained code, with the help of some programming support files

Most of the details (inclusive of the server implementation) are kept in two separate paper documents. We recall here only some basic facts and give some idioms about the construction of graphics clients.

Overwiew of graphics primitives

This list summarized the graphics primitives, with the opcode and operand sent by clients (typically by `graphserv y_*` routines) to servers. Operations fully implemented are listed in **green** (with the link to the relevant routine). Operations not implemented at all are listed in **red**. Operations implemented at server level but without corresponding routine are listed in **yellow** or **orange** (the latter are implemented only in the

PostScript server). An asterisk [*] marks those function unimplemented (no-operations) as meaningless for the PostScript server.

opcode	function	operands
none	**Reconnect** implemented at server level	
-2	**server specific directives**	subcode suboperands
-1	**Terminate** implemented in `deleteserver`	
0	**Disconnect** see y_closeplot	
1	**ClearPage** see y_page	
2	**Move** see y_move	x y
3	**Draw** see y_draw	x y
4	**PolyLine** see y_lines	n x(n) y(n)
5	**PolyMarker** (psserver side only)	n x(n) y(n)
6	**PolyFill** see y_fill	n x(n) y(n)
7	**Text** see y_text	nchar string(nchar)
8	**ClearView** see y_clear_viewport	
9	**WriteImage** see y_write_image	nbyte data(nbyte)
10	**WriteLut** see y_writelut	start ncolor red(ncolor) green(ncolor) blue(ncolor)
11	**ReadCursor** see y_get_cursor [*]	none ; **returns** x y key
12	**ReadImage** [*]	none ; **returns** nbyte data(nbyte)
13	**ReadLut** see y_readlut	none ; **returns** start ncolor red(ncolor) green(ncolor) blue(ncolor)
101	**Viewport** see y_viewport	xlow xup ylow yup
102	**Window** see y_window	xlow xup ylow yup
103	**Coord** see y_coordinates and y_window	ncoord
104	**Scale** see y_scale	xscale yscale
105	**Pen** see y_colour	colour
106	**LineWidth** see y_width	width
107	**LineStyle**	style + TBD
108	**Bkg**	colour
109	**Marker** (psserver side only)	n
110	**TextFont** see y_text	n
111	**TextSize** (psserver side only)	size
112	**TextOrient** (psserver side only)	angle
113	**TextMode** (psserver side only)	n
201	**QueryViewport**	**returns** xlow xup ylow yup
202	**QueryWindow**	**returns** xlow xup ylow yup
203	**QueryCoord**	**returns** ncoord
204	**QueryScale**	**returns** xscale yscale
205	**QueryPen**	**returns** colour
206	**QueryLineWidth>**	**returns** width
207	**QueryLineStyle**	**returns** style + TBD
208	**QueryBkg**	**returns** colour
209	**QueryMarker**	**returns** n
210	**QueryTextFont**	**returns** n
211	**QueryTextSize**	**returns** size
212	**QueryTextOrient**	**returns** angle
213	**QueryTextMode**	**returns** n

Typical graphics clients

- Typical very high level graphics clients in XAS are the display and overtrace commands. These programs do not do any plotting themselves, but are just dispatchers to other clients. Dispatchers are explained in section 4.
- Typical high level graphics clients (used to plot a XAS data file in a standard way) use xasgraph utility routines and have a basic structure like this :

`CALL CHECK_OVERTRACE(CLEAR)`	verify if plotting over preexisting plot
get program specific options	

open and read data file	
CALL GET_DATASTYLE(ISTYLE)	retrieve plotting style from environment
LUS(1)=1 LUS(2)=2 CALL Y_OPENPLOT(LUS,' ',0)	open plotting connection
CALL DF_PEN_COLOURS(LUS,CLEAR)	take pen colours from environment
CALL DF_VIEWPORT(LUS,BUFFER,ANNOTATE)	arrange default viewport
CALL DF_WINDOW(LUS,CLEAR ,REPLOTAXES, ...)	arrange default window

IF(CLEAR.OR.REPLOTAXES)THEN
CALL DF_AXES(LUS, XLINLOG, YLINLOG, ...)
ENDIF
	plot axis frame

.... CALL Y_COORDINATES(LUS,2)	use world coordinates
CALL PLOT_XXY_*xxx*(LUS, ...)	use high level call to plot data
.... CALL Y_COORDINATES(LUS,1)	use NDC coordinates
CALL Y_VIEWPORT(LUS,0.0,1.0,0.0,0.5)	viewport for annotations
CALL Y_COLOUR(LUS,PENCOMMON_TXT)	pen colour for annotations
CALL ANNOTATE_NEW(LUS, ...)	custom annotations
IF (CLEAR) CALL Z_SET_GLOBAL ('LASTPLOT', 'spectrum'//PLOTTYPE//' '//NAME1)	save plot type to environment
CALL Y_CLOSEPLOT(LUS)	close plotting connection

- Plain lower level graphics clients use graphserv routines directly and may have a basic structure like this :

LUS(1)=1 LUS(2)=2 CALL Y_OPENPLOT(LUS,' ',0)	open plotting connection
CALL Y_VIEWPORT(LUS,0.0,1.0,0.0,0.5)	set viewport
CALL Y_WINDOW(LUS,XVL,XVU,YVL,YVU)	set plotting window
CALL Y_COORDINATES(LUS,2)	use world coordinates
CALL Y_SCALE(LUS,'LIN','LOG')	set axis scales
.... CALL Y_COLOUR(LUS,ICOL)	set pen colour
CALL Y_LINES(LUs,2,AX,AY)	use low level call to plot data
.... CALL Y_CLOSEPLOT(LUS)	close plotting connection

- Programs needing graphics input may have a basic structure like this. The trick with DF_VIEWPORT, DF_WINDOW (with dummy arguments) and the xyaxis environment variable is to be regarded as a workaround to make sure this program uses the last axis setting available in the server and in the environment (in a future the "query calls" will allow to get the data directly from the server) and therefore returns consistent coordinates.

LUS(1)=1 LUS(2)=2 CALL Y_OPENPLOT(LUG,' ',0)	open plotting connection
CALL DF_VIEWPORT(LUG,BUFFER,BUFFER2)	recover current viewport
CALL DF_WINDOW(LUG,.FALSE.,.FALSE. ,BUFFER, BUFFER ,BUFFER2, E,E,E,E,E,E)	recover plotting window
CALL Z_GET_GLOBAL('XYAXIS',BUFFER) READ(BUFFER,*,IOSTAT=IERR)X1,X2,Y1,Y2	retrieve actual window extrema
.... CALL Y_GET_CURSOR(LUG,A1,E1,BUFFER(1:1)) CALL Y_GET_CURSOR(LUG,A2,E2,BUFFER(1:1))	read two cursor points
.... CALL Y_CLOSEPLOT(LUG)	close plotting connection
.... IF(E2.LT.E1)THEN ...	better check points are in order

[Previous][Next] [Up][Down]

12. XAS file format reference

XAS uses a consistent, mission-independent private file format for its reduced data files. The complete reference for the XAS file format is reported in two separate paper documents. We give here only an overview of the main characteristics.

- XAS uses only two types of files (and of associated Fortran i/o) : plain ASCII files with sequential access for small quantities of ancillary information, and fixed-format binary files with direct access for real and bulky data.
  A (Unix-like) STREAM_LF format is preferred for ASCII files on non Unix systems (and imposed at z_open level).
  Some additiona support is provided for ASCII tables
  The remainder of this specification covers exclusively the binary XAS data files.

- XAS binary files use the native binary representation of the operating system (OS) on which they are written.
  There is no requirements of portability of the files (one cannot read a file written on a different, uncompatible OS), but there is an interoperability requirement. The OS is identified, and an in place conversion to the local OS format (localization) is supported by a dedicated program and routines. This conversion is unidirectionally supported : a file can be imported to the local OS, but not exported to a generic OS.

- A XAS file is composed in this order by these three parts :
  - a mini-header covering one or more record with 28 bytes of useful information (more than one record only when the record length is less than 28 bytes, the rest being null-padded).
    - a 16-byte magic number in the form $\mathtt{XAS}_1\mathit{PPP}_2\mathit{TTT}_3\mathit{SSS}_4$, where the 1..4 values at subscript position indicate binary values 1..4 (CHAR(1) to CHAR(4)), while the ASCII sequences *PPP* and *TTT* corresponds to types and subtypes described below, and *SSS* is the three-letter code of the operating system which wrote the file (as returned by z_op_sys, one of those stored in the SYSCOMMON common block). The mixture of ASCII and binary code is used to make unlikely that the magic number pattern be generated by chance.
    - an INTEGER RECLEN, the file record length in bytes
    - an INTEGER DATASIZE, the number of data records
    - an INTEGER HDRSIZE, the number of header records containing the header keywords (this does not include the mini-header)
  - DATASIZE data records
  - HDRSIZE header records

- There are two basic layouts for data records (images and [binary] tables) which share a common layout for header records.

- **image** files (magic number $\mathtt{XAS}_1\mathtt{IMG}_2\mathtt{FLO}_3\mathit{SSS}_4$) store an image(nx,ny) in ny records of record length nx*4 (in the default format of REAL*4 images). INTEGER*2 images and 3-d images are supported at specification level, but their usage is not incouraged nor supported by s/w.

- **response matrices** files (magic number of the main file $\mathtt{XAS}_1\mathtt{IMG}_2\mathtt{MAT}_3\mathit{SSS}_4$) are stored as a couple of images, a main matrix M(ne,nchan) in units of cm^2keV (product of RMF * ARF * width of energy grid bins) and an associated histogram H(ne,1) with the input energy grid

- all other data files are stored as XAS binary tables

- a **spectrum** (magic number of $\mathtt{XAS}_1\mathtt{BIN}_2\mathtt{SPE}_3\mathit{SSS}_4$) has 4 REAL columns, namely
  1. `LOWER BOUNDARY`
  2. `UPPER BOUNDARY`
  3. `DATA`
  4. `ERROR`
  5. additional columns may follow for particular, unofficially supported cases (storing fitted models or photon spectra)

- a **time profile** (magic number of $\mathtt{XAS}_1\mathtt{BIN}_2\mathtt{TIM}_3\mathit{SSS}_4$) has n records (as many as n time bins), with a record length such to contain a selection of some of the following columns (when they appear, they do in this order, but what matter is the logical name ; those which do not appear are constructed from header keywords)
  1. `TIME` : the bin start time in seconds since a reference Unix time `TIMEREF` stored in the header, usually the 0 UT of the day containing the first data point in the observation, stored as a DOUBLE PRECISION value. May be absent for fully equispaced time profiles.
  2. `BINSIZE` : a REAL number indicating the bin duration. May be absent for time profiles with all equal bins (refer to keyword `BINSIZE`
  3. `DEADTIME` : a REAL 0.0-1.0 number giving the coverage fraction of the bin (optional)
  4. `DATA` : one value or an array of REAL data values (may be absent for the rare case file of times or time intervals)
  5. `ERROR` : the associate errors (optional)
  6. additional data and error columns may follow to allow parallel storage of more light curves with same time binning (but this has no support in s/w).

- for unofficially supported folded light curves TIME may be replaced by PHASE

- a **photon list** (magic number of $\mathtt{XAS}_1\mathtt{BIN}_2\mathtt{PHO}_3\mathit{SSS}_4$) has has n records (as many as n photons or events), with a record length such to contain a selection of data columns corresponding to the data fields selected for each event.

- any other unspecified tabular file has a magic number of $\mathtt{XAS}_1\mathtt{BIN}_2\mathtt{GEN}_3\mathit{sss}_4$)
- the file header is located at the end of the file in order to be extendable (one can add new keywords) without changing the data part of the file.
- the file header is a sequence of binary-encoded keywords, stored spanning records of the same natural size used for the data. Records are null-padded which allow implicitly to generate an end-of-file keyword (this is not necessary if real keywords fill completely the header space).
  keyword handling routines read the entire header in memory and operate on such memory copy.

- keywords have a n 8-character NAME, a datatype and a length which is encoded together with the keyword value(s) in a binary data structure like this

data type	1 byte type	1 byte length	8 bytes	length bytes
INTEGER*2	1	2*n	NAME	n=l/2 INTEGER*2 values
INTEGER*4	2	4*n	NAME	n=l/4 INTEGER values
REAL*4	3	4*n	NAME	n=l/4 REAL values
DOUBLE PRECISION	4	8*n	NAME	n=l/8 DOUBLE PRECISION values
CHARACTER	0	2<=n<=68	NAME	one CHAR*n string
end of file	0	0	absent	absent

  More data types with particular semantics (angle, time, date) are foreseen by the specifications but not implemented. The usage of the INTEGER*2 type, although defined, is deprecated.
  The keyword data structure can in principle reach n=255 bytes, but for reasons of FITS compatibility is limited to 68 byte character keywords (and the same limit imposed to other data types).

- There is a small number of mandatory keywords which must be present in a file in order for it to be usable. These are usually flushed to disk by the file creation routine, so that a file is usable even if no further ("documentation") keywords are added.
  The number of mandatory routines is much less than in a FITS file since some information is not useful or generated implicitly.
- For an image only BITPIX,NAXIS1 and NAXIS2 are mandatory (NAXIS if image is not 2-d)
- For a table BITPIX=8 is included for documentation, while NAXIS1, NAXIS2 and TFIELDS are mandatory, together with the TFORMn of all present columns, and the TTYPEn names of all used columns. The distinction between present and used columns is due to the need of table padding of record length to a multiple of 4.

[Previous][Next] [Up][Down]
