Solicited talk at the 5th Intnl. Workshop on 'Data Analysis in Astronomy',
Erice, Oct 27-Nov 3 1996, to be published by World Scientific Publishing Corp.

THE *XAS* DATA ANALYSIS SYSTEM

L.CHIAPPETTI
Istituto di Fisica Cosmica (IFCTR), CNR, via Bassini 15, I-20133 Milano, Italy

D.DAL FIUME
Istituto TeSRE, CNR, via Gobetti 101, I-40129 Bologna, Italy

The X-ray astronomy Analysis System (XAS) has been developed as a prototype software in view of the SAX mission . In this paper a description of the XAS architecture and implementation is given.

1 Introduction and history

The X-ray astronomy Analysis System (XAS) is a software package developed in the framework of the SAX mission,[1] with particular regard to the support of the Italian Narrow Field Instruments.[2-4]

Discussions about analysis software for SAX started within SAX Consortium Institutes since 1985 (with a very inhomogeneous computer base, from IBM mainframes to HP minis and early VAXes). A request was soon placed to the Italian Space Agency for procurement of common (VAX) hardware, but the need for portability to Unix (unknown to most of us until 1990) was already a clear design goal. Most of the early design was done during 1988-90, resulting in an ambitious *Proposal for XAS specifications* (the "*XAS dream book*"). This idea for an overall, mission independent X-ray astronomy reduction *and* analysis package was soon descoped due to chronical lack of dedicated manpower, although the basic guidelines (see Sec. 2) were preserved. Coding of the system-dependent library started in 1991, and a first prototype of XAS was available internally at the end of 1992. The work continued since then intermittently, compatibly with our activities within and outside SAX, and with very limited manpower. XAS-compatible *ad-hoc* programs have been used within SAX Italian Institutes for the analysis of ground calibrations since 1994. A central "compensation chamber" for maintenance of a SAX software archive (at SAX SDC[5]) has been created only in late 1995. During 1996 XAS was tailored to real flight data, and final adjustments are in progress.

2 Design guidelines

Portability at least to VMS and Unix has been achieved isolating system dependent features in a small number of routines in a single library (the VOS library, Sec.3.4). We also chose to support only the minimum number of features really needed (a "reduced instruction VOS"). This approach proved successful and 93% of the code is system independent (actually for a long time we kept a single physical set of master sources on a VMS disk NFS-mounted from Ultrix and Sun machines).

Ease of installation implies that no system privileges (any user can install XAS) and no excessive disk space are required , there are no excessive or unusual resource requirements, and there is a simple directory organization (Sec. 3.5).

We wanted the software to be *under full control of the developers*, therefore we use our own XAS file format (Sec. 3.7), do not depend on commercial libraries (but use only home-grown or public domain code), do not use any overarching externally-designed environment.

Also for *ease of programming* we adopted a layered design of library routines, isolated specific competences (e.g Xlib or Postscript graphics programming, or SAX instrument knowledge) in single modules (allowing in principle to have separate persons doing independent development), and, chiefly, coded almost entirely in Fortran (the language of election in our scientific community). Actually only 5% of our Unix code (and less than 1% of our VMS code) is written in C (and with the exception of one Xlib front-end this is all concentrated in VOS "jacket" routines).

Efficiency has been achieved using small, layered, library routines, using native internal representation in the XAS file format, making use of dynamical memory allocation, adopting a client-server approach to graphics (Sec. 3.8).

Ease of use is achieved by a consistent and versatile user interface (supporting command line, interactive query-and-answer and parameter files), avoiding the use of an overarching monitor or shell, allowing access to system environment variables, and replacing simple procedures by front-end dispatcher programs.

Mission independence is achieved using an ad-hoc XAS file format for reduced data products, accumulated as soon as possible from the mission-dependent telemetry, and using environment variables and file keywords to control mission- and instrument-specific items.

We also wished XAS to be *open* to users' favourite analysis packages, therefore the XAS file format was designed to be easily mappable to FITS (for easy exchange/export to/with major packages). XAS files are immediately usable from IDL or SAOImage. Interface problems are isolated in specific conversion modules (to/from plain and OGIP flavours of FITS), which allow to simplify maintenance.

3 Architecture

3.1 User interface

In XAS there is no monolithic data pipeline, and no overarching monitor program, but simple standalone programs which are called on the command line. Command procedures (scripts) are left at host shell level (according to users' preferences).

Each command can be invoked in a fully interactive manner (just issuing the program name), in which case the user will be prompted for all necessary arguments. Otherwise all arguments can be passed on the command line, e.g. as in

accumulate hk *file parameter pkt yy mm dd hh mm ss yy mm dd hh mm ss binsize*

or some arguments only are passed on the command line (those missing will be prompted). Runstring arguments are usually echoed simulating query-and-answer dialogue. Finally one can put all or part of the arguments in a command file (one query per line) and use a global variable (see Sec. 3.2) to signal the program to use the command file instead of the terminal for input. Any argument present on the run string has precedence over command file, and is also possible to force a command file to always prompt interactively for a specific argument, using an escape character.. Command files are useful as standard setup parameter files, e.g. one can do

```
xasset command mysetup
accumulate hk myfile ,, m1dir002
```

in order to accumulate an HK profile myfile from MECS unit 1 direct packets, using standard setup with 60 sec binsize, for counts above minimum energy, where mysetup.command is

```
!               ! filename (escape to terminal query)
valemin         ! parameter name
!               ! packet type (escape to terminal query)
,,,,,,                  ! start time = default
,,,,,,                  ! end time = default
60              ! binsize
```

The global variable command is peculiar as it retains its value only across the execution of the next program. It can however be forced to permanent (e.g to analyse different datasets with a common setup).

It shall be noted also that a command may actually run different programs (this uses a VOS call; Sec. 3.4). This is used (instead of system-dependent scripts) as a mechanism to implement dispatchers. E.g. a command like accumulate hk or accumulate image etc. will actually run a program saxhkaccum or saxiaccum etc. dedicated to the accumulation of HK time profiles, images, etc. Or a command like display *file* will run a program splot, tplot or idisp to display graphically a spectrum, time profile or image, sensing the type of file (from the extension or from environment settings). This also allows to keep an English-like syntax for commands.

3.2 The environment

Communication between the user and the programs run by hir (other than program arguments described above), as well as between programs run in sequence, is obtained using *global variables*. XAS global variables are implemented as VMS global symbols, or Unix environment variables. The collection of XAS and non-XAS global variables is termed *the environment*.

XAS global variables are set either programmatically via a VOS call or by the user via the dedicated command xasset *variable value*. Some (graphics-related) variables whose value is a comma-separated list of items are set via command xasplot (e.g. xasplot xaxis 0.1 10). One should always use xasset (or xasunset) to handle XAS variables and operating system commands to use non-XAS variables.

The two environments (XAS and system) shall coexist and be disjoinct (each one shall appear as a readonly environment to the other), so that XAS program may access also the system environment and shell scripts may access the XAS environment. XAS global variables have a system name obtained prefixing XAS_ to the XAS name (i.e. xasset pinco panco will set environment variable XAS_PINCO). A query of the value of a non-existing XAS variable may fall back to the value of a system variable of the same (unprefixed) name if one exists. Coexistence of both environments is easy in VMS, while in Unix there might be problems since children inherit the environment from parents but do not pass back any modified environment. This is overcome saving the XAS environment to an intermediate file, which is linked to the duration of the login session. A dedicated command is needed in Unix to import back the XAS environment to scripts wishing to use it.

Global variables are extensively used to control and tune the behaviour of programs (alter default) in all cases in which this is done infrequently and it would not be worth cluttering the command line with extra arguments. Some examples are: to set the "chat level" of some programs (quiet); to define a default data type (images, spectra etc.) to be assumed by following commands (context); to set paths for datafiles (Sec. 3.3); to enable or disable HK conversion to engineering units (hkconvert); to apply time windows or corrections (and relevant options) to accumulations; to control axis, scaling, colours, style etc. of graphics

A list of all global variables and their usage will be placed in the XAS HTML help files (preliminary version at http://sax.ifctr.mi.cnr.it/Xashelp).

3.3 Data directory organization

A XAS user does not have to type long path names, but usually needs to indicate just a filename. The system supplies the filetype (e.g. .image, .spectrum etc.) and the directory path according to context and environment information. Nevertheless the user is able to create files elsewhere than in the current working directory.

This is done via a set of global variables (Sec. 3.2), which define : a virtual root (rootdir); a directory for FOT telemetry files (fotdir); a directory for reduced data files (datadir); a directory for "printouts" (printdir). In addition one can specify a substructure by target, observation date and instrument, and select the order in which the latter information are arranged in a hierarchy. A BUILDPATH routine takes care of creating the proper path (e.g. /re/Monitor/CygX1/Sep12/tpm1.time).

The same routine is also used to locate calibration files in the instrument specific directory (e.g. $XASTOP/calib/sax/mecs/mecs.pcf) allowing portable programs not to be affected by system-dependent filenames. The mycaldir environment variable allows the user to use private calibration files in preference of the XAS-supplied ones.

3.4 The VOS (Virtual Operating System) library

The VOS library isolates all system dependent code. VOS routines (whose name starts with Z_) have the same calling interface on all systems.

On VMS systems all of them (with the exception of a couple of memory allocation/deallocation routines which mimic the Unix ones) are written in Fortran and call directly VMS system or library services. Auxiliary routines are hidden to the user and located in the same source code file.

On Unix systems top level routines are written in Fortran. Most of them call C "jacket" routines (zc_), whenever possible POSIX-compliant, which take care of all adaptions (conversion of exotic argument types, linking system calls when they are not Fortran callable, ...). The differences between different flavours of Unix is confined in about 10 routines.

The following is the list of functionalities implemented at VOS level.

In the *file access* area (see also Sec. 3.6), there are calls to: open a file (replacing Fortran OPEN); return file information (replacing/supplementing Fortran INQUIRE); converting VOS file names to system-dependent names; delete a file ; rename a file.

In the *environment access* area (Sec. 3.2), there are calls to: retrieve the run string; return or set a global variable ; terminate program passing back a status code.

In the *inter-process operation* area there are calls to: run a program concatenating it to the current process (in VMS activating another image in the same process, in Unix execvp'ing a command); spawn a program as a separate process (subprocess in VMS) which then runs independently ; create, open and test a communication channel among two programs ; perform i/o across it. Communication channels are implemented in VMS as mailboxes, and in Unix as named pipes.

Some *miscellaneous utilities* are available to : allocate and deallocate memory (in a Fortran-friendly way) ; detect a control-C interrupt to handle "gracious termination" ; convert REAL and DOUBLE floating data (IEEE to/from VMS).

It is also possible to do some *miscellaneous queries* like : return the current operating system; return the host name; return current terminal device name and size (rows and columns); return user name (userid and full name) ; return current system time; return time current login session started.

Since August 1996 the current development system for XAS is Digital Unix (formerly OSF/1); formerly the VOS library had been fully developed in parallel under VMS and Ultrix. Support to XAS on VAXes is now somewhat limited due to the discontinuation of hardware maintenance of such machines at our Institutes Unsupported ports to SunOS 4.1.X (at IFCTR Milano and later at SAX SDC) and to HP-UX (at ITESRE Bologna) are also available. Port to AXP VMS on Alpha platform has been investigated, and should involve limited changes. However, since there is no availability of such hardware, no work in this area is planned. For similar reasons no work on a port to Solaris 2 on Sun platforms is planned.

3.5 Software directory organization

The following diagram illustrates the directory tree of the XAS software distribution.

The config directory is present only on Unix (see Sec. 4). The top directory is site-dependent and shall be stored in a (system) global variable XASTOP. Italics indicate directories whose content is system-dependent (but *inst*lib indicates a multiplicity of instrument specific libraries (mecslib, pdslib, hpgslib).

```
$XASTOP____________ bin                        program executables
          |________calib ......                calibration file sub-tree
          |______ config ......                xasbuild sub-tree
          |________ doc                        bug list and misc doc
          |_____external _________ fitsio      external library sources
          |______include                       Fortran include files
          |__________lib                       relocatable libraries
          |___ libsource _________ fotlib      library sources   SAX tlm
          |                |______ general                       utilities
          |                |____graphserv               low level graphics
          |                |_______ instlib             instrument specific
          |                |____ xasgraph               high level graphics
          |                |_______ xaslib              XAS file handling
          |________local                       for local customization
          |______source                        main program sources
          |________ vos                        VOS library source
```

The total size of XAS sources is some 25000 lines of code (excluding comment), or about 2 Mbyte with comments. The size of the binaries is system dependent (e.g. 10 Mbyte for OSF but 40 for Ultrix) according to loader details.

3.6 Input-output

XAS deals with two basic kinds of file : binary data files and auxiliary/ancillary files.

Binary data files are intended for Fortran DIRECT access and include SAX-specific telemetry files (FOT files), and mission-independent reduced data files (XAS files).

Auxiliary and ancillary files are plain ASCII files, intended for sequential access. For portability reasons they are implemented on all operating systems as newline-terminated files (STREAM_LF in VMS parliance), allowing NFS access inclusive of the VMS<->Unix case.

A VOS routine layer allows generic file access using Fortran in a portable manner with a minimum of change to usual coding practice (the OPEN and INQUIRE calls are replaced, but normal READ and WRITE are used).

An exception to the above restriction (direct access unformatted, and sequential formatted access only) is made only for the interprocess communication across a channel (Sec. 3.4) which occurs on the operating systems which supports it using Fortran unformatted sequential access, and otherwise through a layer of C routines.

3.7 XAS file formats

XAS reduced data files are binary files in native machine representation, sharing a common design (all files are direct access with a natural record length corresponding e.g. to the image or table width, and a number of records corresponding to the number of image or table rows ; also all files share the same kind of header).

Two main classes of data files exist : image format data include images, pseudo-images (any 2-d equispaced data) and response matrices ; tabular format data include spectra, time profiles (both count rate and housekeeping), photon lists and any generic table.

XAS files are logically divided into a data part (organized as described above) and a header. The header is logically made of a number of keywords, which can be numeric- or string-valued. Keyword naming is consistent with FITS (8 character names etc.), and so is the maximum allowed length of a string value. Numeric data are however kept in binary representation, and a keyword value may also be an array of values. The entire header is read in memory with one operation.

For practical reasons the header proper is located at the end of the file, allowing this to be freely extensible (e.g. to add HISTORY or comments after file manipulation). A mini-header of no less than 28 bytes is located in the first record(s) of the file and contains basic information (a magic number, the record length and the number of data and header records).

A layer of system-independent routines (above the VOS i/o ones, Sec. 3.4) allows basic access to XAS files, and handling of header keywords. An higher layer allows handling of specific data classes (images, spectra, time profiles, etc.)

XAS files can be used as they are from IDL (see Sec. 6) or SAOimage (a saodisp command constructs the adequate runstring), or can be exported to MIDAS, IRAF,XSPEC after conversion to FITS. An additional utility (localize) allows to convert XAS files among different operating system internal representations.

A "XAS ASCII" file format (defined as tabular data preceded by a specially-formatted "magic" header) is used for some calibration files and for access via IDL add-on software (see Sec. 6).

3.8 Graphics

The design goal for XAS graphics was to provide a limited set of simple utilities able to produce standard quick look plots, not publication quality graphics, which is left to each user's favourite tool (e.g. IDL, SAOimage, MIDAS, QDP, etc.).

From an user perspective all graphics occur via two commands, display and overtrace, followed by a XAS file name (be it of type image, spectrum or time profile). The former command draws on a fresh frame, the latter overplots on an existing one. These commands are just dispatchers to the appropriate filetype-dependent utility. Additional programs allow graphic input (e.g. to use the mouse to define time or intensity windows).

From a programmer perspective any graphics (client) program uses high level calls (axes, labels, error bars ...) from the xasgraph library, and lower level calls from the graphserv library. These lower level calls correspond to a small set of graphics primitives (move, draw, plot text, plot image ...) which are all implemented sending a device-independent message (opcode + operands) across a communication channel (Sec. 3.4). Cursor readout is returned on a separate communication channel.

On the other side of the communication channel there is a graphic server, which handles all device dependent chores. In particular we have adopted the principle "*let the server do the scaling*" among world, normalized and device coordinates.

There are two implementation of a graphic server, one for X11 and one for Postscript. More than one instance of a given type of server can be started (via the createserver command), and addressed separately using the plotter global variable.

The X11 server uses Xlib, however it is written almost entirely in Fortran, and handles each primitive loading the necessary information in a common block, and making an argument-less call to a C routine afterwards. These C routines are all entry points in a single f2x.c source (less than 400 lines of code), which has global access to the common block as an external struct, and does the Xlib calls.

The Postscript server exists in three flavours (black&white, colour, and colour Level 2), which are actually handled by a single program issuing the same Postscript macros, while all differences between flavours are handled loading a different prologue file with macro definitions.

4 Development tools

While day-to-day development of XAS was done according to taste of the authors, the Unix version of XAS includes xasbuild, a front-end used to generate and mantain Makefiles. It is a collection of shell scripts which allows to generate dependencies for each module (this relies on an analysis of the linker map, and is currently available only under Ultrix and OSF/1), to create makefiles from prototypes and to mantain or archive "configurations". If suitable xasbuild files are distributed along with a partial software update, it will be possible to recreate the full set of makefiles for local use.

The way dependencies are arranged is somewhat unusual : a program or routine source depends on the include files it refers; a library depends on the routines it contains; a main program executables depends only on the sources of all the routines it calls and not on the library (therefore if a routine is updated, only the programs actually calling it are re-made); object files are ignored.

The way the Fortran compiler is invoked by xasbuild is such to resolve correctly include statements (since they contain file names they are usually system dependent), which shall be of the form INCLUDE 'filename.inc' (pathless and lower case). All include files are located in $XASTOP/include.

5 SAX analysis data flow

The starting point for SAX data analysis are the Final Observation Tapes (FOTs), the medium on which SAX telemetry files (with a minimum of reformatting) and auxiliary files are distributed to observers.[6] Tape files related to one pointing (an *Observing Period*, divided into *observations* with a given instrument configuration) are kept in a fixed-blocked format. The first step is therefore *FOT filing*. The program fotfile is somewhat unusual since it solves the problems of system-dependencies of different tape drives just demanding this to external utilities (for Unix the mt and dd commands). The program actually runs as a chain of steps in which a Fortran program writes a script, passes control to it, which in turn passes control to the next Fortran step, etc.

The next step (for each instrument) is the inspection of the *instrument configuration*, a summary of which is produced by check_expconf. As a result of this one defines in the environment which are the observations to be chained (with concatenate). Different chains may be used for science or housekeeping reduction.

The accumulate front-end program allows to produce (from the FOT files) system-independent data structures like images, spectra, time profiles or photon lists. In principle system-independent *cross-accumulation* (xaccumulate) programs allow to do the same from XAS photon lists. Usually we find more efficient and/or appropriate to accumulate straight from FOT files, including on the fly - when enabled - all necessary instrument-specific corrections[2-4] (spatial, energy and time dependent corrections, application of time windows, selections).

The description of the telemetry format is kept in packetcap files (modelled after Unix termcap or printcap), so that support to a given format can be added without recompilation (we took advantage of this to support peculiar ground calibration formats), since most of the code is independent from the original telemetry format.

Other programs allow graphical presentation of data (Sec. 3.8) and cursor-assisted generation of time and intensity windows. One can therefore iterate accumulations after selecting the best time, energy and position ranges. It is also possible to use IDL or SAOimage at this stage, as well as to export XAS files to FITS.

As a particular kind of accumulation the accumulate matrix front-end allows the generation of instrument response matrices in XAS and OGIP formats.

6 Add-on software

In addition to the minimal set of software necessary to reduce SAX data, other software related to XAS (which should however be considered no more than unsupported contributions) includes : EPOS, simulation software for EPIC using XAS library and files; xasread, elementary utilities to access XAS files from IDL; xasplot, a widget-oriented IDL front-end to XAS data display; some add-on pre-existing spectral and timing analysis ported to XAS compatibility.

Acknowledgments

A key role in the early design was played by M. Morini (at the time at IFCAI). The following persons contributed to single XAS modules or instrument specific modules: F.Giambertone, T.Mineo, A.Santangelo, G.Fazio (IFCAI), M.Orlandini. L.Nicastro (ITESRE, the latter also for the IDL contributed software xasplot), F.Fiore, S.Signorile and M.Guainazzi (SAX SDC, the latter in particular for overall maintenance of the SAX software archive). Finally thanks are due the SAX Data Analysis Working Group (DAWG) chaired by M.C.Maccarone since 1991.

References

1. L.Scarsi, "The SAX Mission", *these proceedings*
2. B.Sacco *et al.* , "The MECS Experiment onboard SAX...", *these proceedings*
3. S.Giarrusso, "The HPGSPC Experiment onboard SAX...",, *these proceedings*
4. D.Dal Fiume *et al.* , "The PDS Experiment onboard SAX...",, *these proceedings*
5. P.Giommi, "The SAX Scientific Data Centre", *these proceedings*
6. L.Chiappetti, Sec. 5.4 of SAX Observers' Handbook 1.0, ed. by L.Piro (1995)