επος

EPOS : **EP**IC **O**bservation **S**imulations

L.Chiappetti - IFCTR

Document history

Issue 2.1	28 Aug 1995	Minor updates in section 2 and appendix A,B
Issue 2	19 May 1994	First configured issue
Issue 1.1	13 Dec 1993	Interim update, not configured
Issue 1.0	23 Nov 1992	Original issue, not configured

A bar on the margin marks paragraphs changed between the current and the previous major issue (1.1 to 2)
A single bar marks minor updates (2 to 2.1)

Table of content

Table of content

References

[1] P.Sarra, "EPICS. Note sui programmi di simulazione", P.Sarra 04/09/91 con aggiunte di N.La Palombara 15/04/92

[2] G.Peres, "X-ray sky simulations and their folding through XMM mirrors" May 31 1991, plus README files on node ASTRPA::

[3] L.Chiappetti, "The Randomizer - a software tool for general purpose simulations", issue 1.1, Nov 93

[4] A.Bouere, C.Cara, M.Lortholary, C.Pigot, Description of the EDU architecture and operation, Issue 0.1 - Nov 1993

[5] L.Chiappetti, "The Real Programmer Tool. A set of procedures for Fortran programmers", Version 1.6 Jun 93

1 Introduction

What is EPOS ?

EPOS is the new package of simulations for EPIC which is being developed at IFCTR, and is going to replace EPICS, the suite of programs originally developed by P.Sarra (reference [1]).

Why EPOS ?

EPOS is being written in a system-independent way :

all dependencies on operating system (VMS) and on external libraries (NAG) present in EPICS are being removed. EPOS program sources at present exist as a single physical copy on the IFCTR Vax disk, and are accessed via NFS to be compiled, linked with the appropriate libraries and run on Vax, Sun and DECstation under VMS, SunOs Unix and Ultrix.

EPOS uses a new format for data files (the XAS format):

this format supports both images and tabular files (mainly photon lists for EPOS, but also spectra and light curves), with a common handling of a kewyord-oriented header. Data are kept in a system native, compact binary format (utilities to convert data representation across systems are planned). The XAS format is not FITS, but can be converted in a straightforward way to FITS, since it has been designed with that in mind.
XAS data files can be read immediately within IDL and SAOImage, or converted to FITS and used with MIDAS and IRAF.

EPOS uses a common, consistent and flexible user interface :

All EPOS programs can be run interactively, or passing arguments on the run string, or using standard command files (which replace the EPICS parameter files), or a whatsoever combination, Some EPOS operation may also be driven by environment global variables.

The development of EPOS programs, the file format and the user interface benefit of the previous work done by the author for XAS. XAS, the X-ray Astronomy analysis System, is a prototype system proposed originally for the SAX mission and being developed privately by the author.

This document is structured in a section (2) describing the various simulation steps, another describing the file formats (3.1) and the standard reference files (3.2), and another with the listing of the standard configuration files (4). Some examples of its usage are reported in section 5. Two appendices give user's and programmer's instructions.

2 Steps in EPOS

EPOS proceeds in a sequence of steps. Some steps are necessary only for a complete simulation, which may be unnecessary for particular purposes, and therefore some steps can be bypassed, approximated, or collectively merged into a single approximation.

A. Input data simulation

The first potential step is the production of a list of photons at the entrance of the telescope. The parameters of a photon will be time T, energy E and two angular coordinates θ,ϕ (off-axis angles from the telescope optical axis). One might want to consider separately the target, the sky background and the particle background :

A.1 Target source simulation

For each target source (generally one, but could be more) one has to simulate at least the energy spectrum *OR* the time behaviour (according to the purpose of the simulation). Typically one will assign all photons to a single celestial position.

A.1.1 (point-like) source spectrum

There are three ways of simulating a source spectrum.

The original Nov 92 EPOS contained no modules doing this for celestial sources, but we relied on files produced by the Observatory of Palermo (OAP, see reference [2]).

However Nov 92 EPOS included an utility program `linefield`, which can be used to produce a list of photons diffused all over the field of view in a single monochromatic line (this is mainly useful for testing and debugging). The syntax is :

```
linefield seed nphotons energy xlow xup ylow yup
```

This creates a file `line.photon` with uniform random positions (in micron), constant energy (keV), and null times. This file may then be renamed as wished.

Nov 93 EPOS is now able to produce a list of photons distributed according to any spectral form described via an external program, using the `randomize` package (see reference [3]). `randomize` has several variants, among which one allows to generate fresh files (with just photon energy), one allows to write energies to an existing file, and one allows to precompute the number of events without generating any file.

The spectral functions will include appropriate reference to the mirror nominal geometric area, held in a reference file `geometric.area`.

For practical reasons it is recommended to proceed as follows :

- first compute the number of events for a given spectral form and exposure time
  `randomize` *`function parameters exposure e1 e2`* `COUNT`
- then use `randxy` (see below B2.1) to generate a file with photon positions
- then use `randomize` again to fill in the energies in the file created by `randxy`

```
randomize function parameters exposure e1 e2 OLDXAS file
scalefactor seed
```

A.1.2 (point-like) source time behaviour

The original Nov 92 EPOS contained no official modules in EPOS doing this. The files produced by OAP contain photons uniformly distributed in time. The modules present in EPICS to perturbate/modulate the arrival times of such photons have not yet been converted to EPOS.

Nov 93 EPOS is now able to produce a list of photons distributed according to any time modulation described via an external program, using the `randomize` package (see ref. [3]). Typically one will use `randomize` to insert times in a pre-existing file (either a celestial point source created by `randxy` and `randomize` spectral version, or a monochromatic source created by `linefield` and `randxy`) using :

```
randomize function parameters t1 t2 OLDXAS file scalefactor
seed
```

A.1.3 extended source shape

At the moment there are no official modules in EPOS simulating an extended source (either realistic or purely geometric). Some work on this is being done at OAP.
The program `randring` allows to simulate a ring-like source (optionally including PSF similarly to what described in B.2.1)

A.2 Sky simulation

A parallel step to the simulation of the target involves the simulation of the sky background. The two components may be followed separately up to the moment of reading out the CCD (step D.2 below).

A.2.1 resolved component

Part of the sky background is due to a collection of remote and close point sources (for which at least a spectrum shall be simulated, time behaviour can be assumed uniform).
We rely at the moment on a single background reference field provided by OAP (see ref. [2]).
However one must note that such file was produced for 1+2*(1/2) mirror modules, and not for a single mirror module.

A.2.2 LogNLogS

A.2.3 Field stars

On both topics see quoted reference [2] from OAP

A.2.4 diffuse background

The residual diffuse sky background could be generated and inserted in the same background reference file containing the resolved component. At the moment we use the quoted file supplied by OAP (reference [2] from OAP)

A.3 Cosmic ray background

In parallel with the generation of the target and sky photons one shall also produce a list of particle events. At the moment this occurs contextually with the simulation of CCD interaction (step D3). There is no "FOV" level particle file.

B. XMM Optics simulation

This process will be applied to target, sky and diffuse bkg files (but not to the particle file) and will convert θ,ϕ into x',y' (linear measure over the focal plane, e.g. mm or µm) leaving the other information (E,T) unchanged. Additionally it might take into account the optics transmission (and reject a percentage of events).

In principle this process can be implemented in an exact or approximate way:

B.1 Full simulation (ray-tracing)

The full simulation will need a ray tracing. The ray tracing program which used to run on the IBM mainframe has currently not been incorporated into EPOS.

B.2 Approximate simulation

The approximate simulation will require a PSF and an optics transmission (they could be applied separately and in any sequence).

Nov 92 EPOS had no modules doing this. Application of a (constant) PSF is already done by OAP when creating the target and sky files (see reference [2]).

However Nov 93 EPOS is now able to perform the two steps of the approximate simulation via two separate commands, the first of which is:

B.2.1 PSF simulation

```
randxy file nevents sigma seed identifier
randxy file sigma seed
```

which either create a new file with the wished number of events, corresponding to a gaussian point source of given `sigma` (use value in arcseconds) located in the centre, or will insert random positions (distributed according to a gaussian of given `sigma`) into an existing file (this is the case of files produced by linefield ; use values in micron). A radially symmetric gaussian is used. The file must not exist in order to use the first form above.

B.2.2 Optics transmission

The second step, to be applied collectively to all cases, is to reject some photons according to the optics transmission. This is handled using the `filter` command described below in C.2 (and the reference file `mirror.transmission`).

B.3 Converting files from OAP

In EPOS Nov 92 processing starts reading an `.oap` ASCII file coming from OAP and converting it into a XAS format `.photon` file. At the same time angular coordinates are converted from OAP units into θ,ϕ (in milliarcsec with origin at the centre of FOV), and all events lying outside the FOV radius are rejected. The syntax is :

```
oapxas filename fovradius identifier
```

B.4 Converting to focal plane positions

Since `oapxas` can handle OAP files both before and after the optics, it generates photon files with angular coordinates. For analogy `randxy` usually creates files in angular coordinates (however there are cases in which microns are used directly). To convert (in place) the coordinates in the created photon files from (milli)arcsec to microns in the focal plane (origin at centre of FOV) assuming a focal length, one uses :

```
focalplane filename focallength
```

C. EPIC filter and dead layer simulation

This needs to be applied to the output of all previous processes and will simply reject events according to a chosen filter transmission. Any dead layer in the CCD itself need to be handled at this level.

C.1 tabulate filter transmission

The `fil_tra` program (originally part of EPICS and acting on `.epic` photon files) has been only slightly modified to provide instead a tabulation of the chosen filter transmission. This program runs on the Vax only (since NAG dependencies implied by the spline interpolation are not removed) using the old EPICS arrangements (parameter files, see reference [1]).

The produced ASCII file *`filtername`*`.transmission` is in `STREAM_LF` format, so that it could be accessed via NFS from Unix. This file is now in "XAS ASCII" format (includes self-documenting file header for IDL usage). In all cases it shall be installed (or soft-linked) in the directory `$XASTOP/calib/xmm/epic`.

C.2 apply filter transmission

The module which performs rejection of photons due to a chosen filter can instead be run on any system, and produces a new photon file containing the transmitted events only. Its syntax is :

```
filter infile outfile filtername seed
```

If `filtername` is just a file name without path, the program will look for it in the place pointed by global variable `MYCALDIR` (if set) or by default in `$XASTOP/calib/xmm/epic1` (for MOS) or `epic3` (for MAXI); otherwise a "customized" filter transmission file can be specified by giving the full path to it. The default file type is `.transmission`. The file must be in XAS ASCII format (use the `xasasc` csh script to convert files imported from elsewhere). The mirror and dead layer transmission are handled identically to filters.

D. Chip-level interactions

Interaction of the photon or particle in the silicon constituting the CCD chips shall be simulated separately for each chip in the focal plane, rejecting preventively all events falling outside the individual chip.

D.1 Converting to chip positions

This module converts micron positions x',y' on the focal plane into pixel positions x,y on a chip of given pixel size and given position (indicated by the location of the chip centre in the focal plane in micron), additionally rejecting all events below a low energy threshold. A set of standard command files shall be created to correspond to the main position of the baseline focal plane arrangement (+ a central position for convenience). The syntax is :

```
chipsize infile outfile pixsize nx ny xc yc identifier lowenergy
```

Note that normally one sets a command file (see section 4 for a list of the standard ones) and just does `chipsize infile outfile`

In the Nov 93 version pixsize may be an array of two values (to allow for MOS store sections with rectangular pixels), i.e. the x and y size. If one only is supplied the pixel is assumed square, for consistency with previous versions.

D.2 CCD interaction simulation for X-rays

The program `ccd_det` uses the original code by C.Pigot and P.Sarra used by EPICS, but reads and writes files according to EPOS conventions. The chip pixel size and dimensions are retrieved from the input file. `ccd_det` will convert E into charge (keeping the same time sequence) simulating the X-ray interaction within the CCD, inclusive of splitting (which may generate additional events around the original x,y). A set of standard command files shall be created with the physical parameters of the EEV, Thomson and MAXI chips. The syntax is :

```
ccd_det infile outfile seed epair fano side depleted fieldfree
        substrate diffusion reflectivity impurity temperature
        splitsize
```

Note that normally one sets a command file (see section 4 for a list of the standard ones) and just does `ccd_det infile outfile`

Dark current is not included in the simulation.

D.3 CCD interaction simulation for particles

The program `cos_ray` was adpated by P.Sarra from original code by C.Pigot and writes *chip-level* files according to EPOS conventions. It offers the choice of three levels of solar activity, and simulates the interaction of protons and alpha particles. The full syntax is :

```
cos_ray outfile seed exptime sunactiv epair fano depleted fieldfree
        substrate diffusion reflectivity impurity temperature
        pixsize nx ny splitsize
```

The syntax is very similar to the one of `ccd_det`, but since there is no preliminary FOV file, one must specify the pixel size and the number of pixels in the chip. The geometric location in the focal plane does not matter (as the current model does **not** take into account shielding by spacecraft structures). To ensure compatibility with steps D1 and D2, one should prepare a software-assisted command file (see section 4 for a list of the standard ones). This is done using the following command prior to `cos_ray` :

```
precosmic     ccdfile chipfile newfile
```

where `ccdfile` is a command file for `ccd_det`, and `chipfile` is a command file for `chipsize`, while `newfile` is the command file created for `cos_ray`. There is a default naming convention, to look for `chipfile` corresponding to chip 1 of the same name of `ccdfile`, and to append `_cosmic` to `ccdfile` for the name of `newfile` (e.g. if `ccdfile=eev`, default `chipfile` will be `eev_chip1` and resulting `newfile eev_cosmic`).

This way afterwards one just does `cos_ray outfile seed exptime sunactiv`

Now `cos_ray` and `precosmic` have been arranged to handle also non-square pixels (for the MOS store section), but the image and store section must be handled separately and merged later (see description in section 5).

E. Splitting, pile-up assessment, statistics et al.

The output of CCD detection steps for photons and cosmic rays shall be merged at least before entering the readout modules. This may take place within the readout simulation process, or for statistical purposes.

E.1 Splitting statistics

The `splitstat` module computes some simple statistics on the result of `ccd_det`. One may apply a charge threshold to ignore all (split and unsplit) events below such threshold. The syntax is :

```
splitstat filename lowthreshold
```

E.2 Charge vs E relation

The `chargevse` module allows to plot the charge vs energy relation. The analysis can be limited to a class of events. This module needs both photon files with energies and charges (i.e. the input and output to `ccd_det`) and produces a third file in output. This file just contains energy and charge for each photon in the class, and is not used for anything else but plotting. The syntax is :

```
chargevse ccddetfile chipfile class lowthreshold outfile
```

Such file can be plotted in IDL as follows :

```
IDL> chargevse,'outfile.photon',energy,charge
IDL> plot,energy,charge,psym=n ....
```

The `class`es are 1 for unsplit events, 2 for central pixel of split events, 3 for reconstruction of split events, 4 for 1+2 and 5 for 1+3.

E.3 Pattern recognition

A statistics of pattern types could be introduced easily in `splitstat`, but it is felt unnecessary, and is deferred to the controller simulation.

E.4 Pile-up analysis

The EPICS pile-up analysis programs have not been currently incorporated in EPOS, since it is felt preferrable to incorporate pile-up analysis in the "dumb" readout simulation.

E.5 Scaler utility

The `scaler` utility shall be actually used before invoking `ccd_det` or `chipsize` (preferably on a focal plane file) to scale the intensity of the flux by a given factor. Scaling is performed by stretching or squeezing the photon arrival times, and is reversible (i.e. if you scale by a factor `f,` and then by a factor `1/f` you obtain the original file), since it occurs in place and the necessary information is written to the file header. The syntax is :

```
scaler filename factor
```

This is typically done on a target reference file. **Beware** that after the scaling the file will cover a shorter or longer time interval, and this typically will be not compatible with the background file. Analysis should be somehow limited to the overlap interval.
Also, since the time is kept as 32-bit number of microseconds, if the stretching will cause overflow, the file will be truncated.

E.6 Shifting (relocating) utility

The `relocate` utility shall be used before invoking `chipsize` to shift all photon positions in x and y by a known amount (This is typically done on a target reference file). The shift is reversible (i.e. if you shift by Δx,Δy, and then by -Δx,-Δy you obtain the original file), since it occurs in place and the necessary information is written to the file header. The syntax is :

```
relocate filename deltax deltay
```

E.7 Merger utility

The `merger` utility is intended to merge three time-sorted photon files into one, copying photons in time order, and flagging them with a pointer to the original file. This program is intended to merge sky background, target and particle background files after CCD detection and prior to readout. It is called as :

```
merger file1 file2 file3 outfile FLAG|NOFLAG
```

The `FLAG` option is currently implemented as follows : the flag information is added to the split-flag : for the first file no addition occurs, for the second file one adds +100, and for the third file +200. Subsequent readout programs rely on the fact that the first file is background, the second one is target and the third is cosmic rays. If any of such files is missing use the keyword `NULL` (all capital) as a placeholder to preserve correct flagging.

F. Readout simulation

There will be a set of processes : at least one for each mode. They will produce a list of x,y,E' (digitized charge) in one or more frames, simulating the output of the readout process to be passed to ECEs.

It is currently planned to have at least two readout simulation processes for each mode : a *smart* one and a *dumb* one.

F.1 Smart readout

Smart readout modules will be based on the conversion of corresponding EPICS modules (not top priority). A smart readout operates on the individual photons and tries to minimize the computing overheads.

F.2 Dumb readout

Dumb readout modules will simulate the readout in a complete way, i.e. put the photon into a 2-d array, and shift it during readout. Shifting could actually be done virtually via a set of indirection pointers. It is easier to analyse pile-up concurrently with readout in dumb mode, as well as to implement CTE effects (this could be incorporated in a second time).

There will be a separate program for each mode, in general with a similar calling syntax (it is intended to provide standard command files - see section 4 - with typical parameters).

`ff_dumb`	Full frame, MOS & pn
`fs_dumb`	Frame store, MOS (shielded) **(a)**
`rfs_dumb`	Refreshed frame store, MOS (shielded)
`w_dumb`	Window, pn and also MOS (generic)
`wfs_dumb`	Window, MOS frame store chips
`fw_dumb`	New Timing (Fast window), MOS
`w2_dumb`	Multi-chip window, pn, low priority **(b)**
`tm_dumb`	Old Timing, MOS **(c)**
`tp_dumb`	Timing, pn
`b_dumb`	Burst, pn

(a) : this program shall no longer be used. One should use `wfs_dumb` instead, using a window covering the entire 600∞600 pixel area. On Unix `fs_dumb` is a front-end script which asks the user which program he really wants to use. The executable of `fs_dumb` is called `fs_dumb_REAL`.
(b) not implemented yet
(c) should not be used any more

Note that the syntax is not identical since some modes may have additional parameters (e.g. window position), and in some modes the choice of integration time is not free. However the typical syntax will be like :

```
x_dumb  infile  outfile  nframes  tic1  [tic2]  [ nodes]   [..]
int_time
```

where it should be possible to confine the readout to a limited number of frames. The two tics generally correspond to the time to shift one pixel with sampling and without sampling, but the exact application within each program may vary.

F.3 Visual readout simulation

If time allows, the dumb readout could be complemented by a visual interface, which displays the image being accumulated (and being shifted !) in an X-windows window. Of course this has a very low priority, since it is essentially of cosmetic nature.

G. Controller simulation

This will operate on the output of the readout step to produce the data to be passed to EDHU. Its main purpose is to simulate the pattern recognition and therefore to generate the "format A1" used by FF,FS and W modes for MOS, etc. etc.
If the particle background files includes also "horizontal tracks" this process shall reject them. However it could be simpler to bypass this at all, and use particle bkg files already without such events.

In Nov 92 EPOS this stage was not implemented pending the numerous uncertainties on the operation of the controller. The module could be replaced by a simple statistics summary, called :

```
pilestat filename lowthr hithr
```

which produces a statistics of unpiled and piled events, sorted by origin (target, background or cosmic) and by type of pileup (see 3.1 7 below). Optionally it could limit the statistics to a range of charges between `lowthr` and `hithr`. After this one could assume a "black box" operation of the controller, with a given rejection percentage for cosmic rays, and/or a given efficiency in reconstructing split events.

G.1. Generic controller

A generic simulation of "ideal" event processing is provided for reference. This handles so far the output of imaging modes and is called as :

```
exce infile outfile lowthr hithr
```

and acts as follows : (a) all cosmic rays, unpiled or piled with other cosmic rays, are rejected; (b) cosmic rays piled with X-rays (and those with flag=99) are kept in further processing; (c) events below `lowthr` are rejected; (d) X-ray unsplit and unpiled events are output unchanged; (e) unpiled split events are reconstructed using flag information and output at the end of each frame, unless they are above `hithr`; (f) all remaining "complex" events, including all those flagged as piled are stored in a frame and processed at the end for a tentative reconstruction, and output, unless they are above `hithr`.
This program was just an attempt and is likely to be of little or no use.

G.2. MOS EDU simulation

G.2.1. Compilation of pattern libraries

The EDU makes use of pattern libraries. A pattern library may be created very simply with an editor in a file *`libraryname`*`.patternlib`. This library needs then to be compiled in a more compact form used by the EDU simulation programs by means of the command

```
compattern libraryname
```

which creates a file *libraryname*.patlib.
The format of both types of files is described below in section 4.6.

G.2.2. Imaging mode

This program simulates the basic mode of the EMCE EDU, and some additional function of the EMDH for what concerns cosmic ray rejection.

```
emce_i infile outfile lowthr hithr patlib edhsim
```

where `lowthr` is the charge lower threshold (use 25 e^-) and `patlib` is the name of a compiled pattern library. These parameters are used by the EDU simulation, which determines whether a pixel is highest in the 3x3 matrix, if it is one of the known patterns, if the event is isolated (i.e. the pattern guard region is empty) and computes the 4 basic charges (central, pattern, residual 3x3, residual 5x5).
As an additional simulation of the EMDH, two sums of the basic charges are *always* produced (best reconstructed energy of photon, total residual energy).
If the flag `edhsim` is `YES`, in addition a check that the photon reconstructed energy is below the high threshold `hithr` (use 2750 e^-) is done (this is an *optional* simulation of an EMDH task) to improve the cosmic ray rejection.

The following effects are ignored or not relevant to the EDU simulation : Gatti number and offset correction, "management of the two sides" across "node borders", bias correction, gain inequality between readout nodes; also coordinates are chip coordinates and not node coordinates.

G.2.3. Timing mode(s)

This program simulates the combination of the timing mode of the EMCE EDU, and of the EMDH (mandatorily for split event reconstruction, optionally for what concerns cosmic ray rejection).

```
emce_t infile outfile lowthr hithr  edhsim
```

where `lowthr` is the charge lower threshold (use 25 e^-), used by the EDU simulation. The default EMDH simulation determines whether a pixel is highest in the 3x1 matrix, if it is one of the four hardoced patterns (single event, double-right, double-left, triple), if the event is isolated (i.e. the pattern guard region is empty) and computes the 4 basic charges (central, pattern, residual 3x1, residual 5x1) for analogy with imaging EDU.
As an additional simulation of the EMDH, two sums of the basic charges are *always* produced (best reconstructed energy of photon, total residual energy).
If the flag `edhsim` is `YES`, in addition a check that the photon reconstructed energy is below the high threshold `hithr` (use 2750 e^-) is done (this is an *optional* simulation of an EMDH task) to improve the cosmic ray rejection.

The following effects are ignored or not relevant to the EDU simulation : Gatti number and offset correction,; also X-coordinates are chip coordinates and not node coordinates (Y-coordinates are time and not spatial).

G.3 pn Event Analyser simulation

G.3.1 Imaging (full frame) mode

This program has to be used after the FF mode (and in the future also after the LW mode). It performs cosmic ray rejection as follows: three entire rows around any event with a charge equal or greater to `maxthr` are totally rejected.

```
epce_i infile outfile lowthr hithr maxthr
```

In practice `maxthr` is fixed to 6793 electrons (i.e. ADC=4096, i.e. 25 keV, that is full scale overflow). In additions events outside the "usual" `lowthr-hithr` range (25-2750 electrons) are also rejected.

G.3.2 Imaging (window) mode

This program has to be used after theW mode. It performs cosmic ray rejection as follows: any entire frame containing an event with a charge equal or greater to `maxthr` is totally rejected.

```
epce_w infile outfile lowthr hithr maxthr
```

The meaning and values of the various thresholds is the same as for `epce_i` (G.3.1).

G.3.3 Generic mode

This program is used in all other modes (T,B) and corresponds to a generic Event Analyser working mode in which just a plain thresholding between the "usual" values is applied.

```
epce_x infile outfile lowthr hithr
```

G.3.4 Split event reconstruction (imaging)

Since the above Event Analyser simulations do not perform any reconstruction of split events, and since a simulation of such reconstruction is needed for the assessment of the limiting rates (irrespective whether this is done on board or on the ground), in absence of better information this is simulated by the following program (which is applicable to all modes producing an imaging output, i.e. FF, W and B).

```
ground_pi infile outfile lowthr hithr patlib edhsim
```

which uses the same algorithm of `emce_i` (note however that `edhsim` should be fixed to `N`, since cosmic ray rjection and effective thresholding have already been done in the EPCE stage. Provisionally the same pattern libraries used for the EMCE are used, but tuning might be required.

G.3.5 Split event reconstruction (1-d)

Similarly to the above one has to treat the 1-d case (T mode) in which spatial information is limited to one dimension. This is done by the following program, which uses the same algorithm of emce_t.

```
ground_pt infile outfile lowthr hithr edhsim
```

H. EDH packing assessment

This will operate on the output of the controller step and will measure the bit rates. It shall not attempt to generate simulated packets at this stage.

In a first instance one could do simple arithmetics on the results of the `pilestat` program, or work elsewhere (e.g. Excel) on counters provided in the file headers.

A few undocumented programs exist and have been used in the assessment for the assignment of some functions to EMCR, EMDH, EPEA and EPDH. These include :

`ndc`	a simulation of the Non Destructive data Compression (MOS)
`rancut`	a simulation of random data selection (for MOS)
`undersample`	a simulation of event undersampling (pn)
`fixtime`	computes the best estimate of event time for the modes which allow it it is required if one wants to derive light curves to be Fourier-analysed

The former two have been used in the simulations described in my memo of 8 Sep 94 ("EMDH related questions").
The latter two have been used in the simulations described in my memo of 1 Jun 95 ("pn undersampling simulations").

3 Types of files

3.1 File formats

All photon files used by EPOS are XAS photon files. They are tabular files (closely resemblant FITS binary table files) with a number of columns. Each column `n` is identified by a format (`TFORMn`, in FITS binary table format), a name (`TTYPEn`), units (`TUNITn`) and a minimum and maximum (`TMINn` and `TMAXn`, the latter provided for convenience, e.g. during image accumulation). Information about the columns can be obtained inspecting the file header. All columns used by EPOS are either `1J` (`INTEGER*4`) or `1I` (`INTEGER*2`), but flag columns may be `1B` (byte). A nameless column of type `1B`, `2B` or `3B` can be added for padding, but is not used by the software.

The header of each file tries to keep track of all processing done on the file, therefore after the basic keywords one has a set of `COMMENTS`, of additional keywords recording parameters of the program (chip characteristics etc., or counters of rejected or accepted events), concluded by an `HISTORY` keyword with the run string used to generate the file. Each further program may add a new set of `COMMENTS`, other keywords and `HISTORY` at the end (as well as modify some of the previous keywords).

3.1.1 Sky files

These are the files produced by the sky simulation or by `oapxas`, or by `randxy` in "new file" mode. See 3.2 for standard reference files.

Column	TTYPE	TUNIT
1	TIME	MICROS
2	THETA	MILLIARCSEC
3	PHI	MILLIARCSEC
4	ENERGY	EV

3.1.2 Focal plane files

These are the same files above files modified (in place !) by `focalplane` or generated by `linefield`.

Column	TTYPE	TUNIT
1	TIME	MICROS
2	X	MICRON
3	Y	MICRON
4	ENERGY	EV

3.1.3 Chip files

These are the files created by `chipsize`. An useful convention is to give to them the same filename of the focal plane file, followed optionally by a chip designator, followed by an `_c` (e.g `bkg_eev1_c.photon`).

Column	TTYPE	TUNIT
1	TIME	MICROS
2	XPOSITN	PIXEL
3	YPOSITN	PIXEL
4	ENERGY	EV

3.1.4 Post-detection files

These are the files created by `ccd_det` and `cos_ray`. An useful convention is to give to them the same filename of the chip file, replacing `_c` by `_d` (e.g `bkg_eev1_d.photon`).

Column	TTYPE	TUNIT
1	TIME	MICROS
2	XPOSITN	PIXEL
3	YPOSITN	PIXEL
4	CHARGE	ELECTRONS
5	FLAG_1	EVENT NUMBER (*)
6	FLAG_2	SPLIT_FLAG (*)

(*) `FLAG_1` is the index of the original photon in the chip file, or the cosmic particle number.
(*) `FLAG_2` is 0 for unsplit events, 1 for the central pixel of split events, 2 elsewhere
for cosmic ray files only values 0 and 2 are used
the `merger` program will modify this flag, adding +100 or +200 for target and cosmic files
therefore the values of the flag in a merged file are 0,1,2 for background photons, 100,101,102 for target photons and 200,202 for cosmic rays.

3.1.5 Charge vs E files

These are the files created by `chargevse` (these files have limited header information and are used as an intermediate file for plotting or generating spectra).

Column	TTYPE	TUNIT
1	ENERGY	EV
2	CHARGE	ELECTRONS

3.1.6 Post-readout files

These are the files created by `xx_dumb`. An useful convention is to give to them the same filename of the `ccd_det` file, replacing `_d` by `_r` eventually with an optional mode designation (e.g `bkg_eev1_rff.photon`).

Column	TTYPE	TUNIT
1	TIME	MICROSEC
2	XPOSITN	PIXEL
3	YPOSITN	PIXEL
4	CHARGE	ELECTRONS
5	FLAG_1	EVENT NUMBER (*)
6 (+)	FLAG_ARRAY	mixed flags

(*) `FLAG_1` is the index of the original photon in the chip file. For piled events it refers to the first photon falling in the given pixel

(+) `FLAG_ARRAY` is a 3-byte field: `FLAG_ARRAY(1)` is the split flag taken from `ccd_det` or `merger` (for piled events it refers to the first photon falling in the given pixel), `FLAG_ARRAY(2)` is the pileup flag, and `FLAG_ARRAY(3)` is the smear flag.

The readout programs used to set the pileup flag adding a +1 for each pileup on the chip, and +10 for each pileup which occurs in the output register (for modes which integrate there). This is no longer true.

The readout programs now reset the pileup flag in an attempt to classify it. This way no counts of individual pileups is provided in the file, nor there is a distinction between on-chip pileups and register pileups. They are only counted at program level, and their total numbers are recorded in the file header.

The following table gives the values of the pileup flag according to the origin of piled events (using the "merged" split flag). The classification is done only if two events pile; if more than two events pile, the flag is set to 99 or "complicated pileup".

Incoming event flag ("split") flag	0	1	2	1 0 0	1 0 1	1 0 2	2 0 0
0 : unsplit background event	3 1	3 1	3 2	1 1	1 1	1 3	4 1
1 : central background event (split)	3 1	3 1	3 2	1 1	1 1	1 3	4 1
2 : residual background event (split)	3 2	3 2	3 3	1 2	1 2	1 4	4 2
100 : unsplit target event	1 1	1 1	1 2	1	1	2	2 1
101 : central target event	1 1	1 1	1 2	1	1	2	2 1
102 : residual target event	1 3	1 3	1 4	2	2	3	2 2
200 (or greater) : cosmic ray	4 1	4 1	4 2	2 1	2 1	2 2	5 1
a any other combination (more than 2events)	9 9	9 9	9 9	9 9	9 9	9 9	9 9

This way the pileflag < 9 indicates a true pileup between source photons which are too close either in space or in time; <19 indicates a coincidence of a source photon with a background photon ; <29 a cosmic ray affecting a source photon ; <39 a pileup between background photons (should be rare indeed); <49 a cosmic ray affecting a background photon and 51 a coincidence of two cosmic ray events. 99 indicates "any other case", typically with more than two events.

3.1.7 Post-controller files

3.1.7.1 Simplified format

These are the files created by `exce`. An useful convention is to give to them the same filename of the post-`readout` file, replacing `_r` by `_e` eventually with an optional mode designation. They have the same format as the post-readout files, but of course contain reconstructed photons, instead of events, and the `FLAG_ARRAY` is changed: the pileup flag `FLAG_ARRAY(2)` is redefined to assume one of the following values :

0	for unsplit unpiled events
1	for split reconstructed events
m=10+n	for complex events (usually resulting from pileup), where n is the number of events being combined into one.

`FLAG_ARRAY(1)` is the split flag, and `FLAG_ARRAY(3)` is the smear flag, and are passed unchanged if the pileup flag is 0 or 1, while they are undefined (and assume the value 255) in the case of complex events.

3.1.7.2 Post-rejection pn format

The output of `epce_i`, `epce_w` and `epce_x` have the same format as the input files (since the programs simply reject presume cosmic ray events), i.e. as described in 3.1.6.

3.1.7.3 Full MOS format

These are the files created by `emce_i` and `emce_t`. An useful convention is to give to them the same filename of the post-`readout` file, replacing `_r` by `_e` eventually with an optional mode designation. They have a format similar to the post-readout files, with the same 6 columns, but with a different dimension:

`CHARGE` is now an array of 6 values :

`CHARGE[1]`	(=`CHARGE[3]`+`CHARGE[4]`) is the event reconstructed charge
`CHARGE[2]`	(=`CHARGE[5]`+`CHARGE[6]`) is the "residual" charge
`CHARGE[3]`	is the charge in the central pixel
`CHARGE[4]`	is the sum of charges above threshold in 3x3 matrix (pattern charge)
`CHARGE[5]`	is the sum of charges below threshold in 3x3 matrix
`CHARGE[6]`	is the residual charge in 5x5 matrix

The last four values correspond to the output of the EMCE, while the former two sums correspond to the result of elaboration within EMDH.

`FLAG_ARRAY` is now an array of 4 values, where the new element `FLAG_ARRAY[4]` is the *pattern number* (1-32 as numbered in the pattern library).

3.1.7.4 Post-reconstruction pn format

These are the files created by `ground_pi`, and `ground_pt`. Since these programs use the same algorithm as the MOS controller simulation, the format of the files is the same as described in 3.1.7.4 (i.e. 6-element `CHARGE` and 4-element `FLAG_ARRAY`).

3.2 Reference files

The original `.oap` files are kept in Unix compressed format (`.oap.Z`) in the directory `/poseidon/lucio/epic/palermo`. The reference `.photon` files mentioned below are focal plane files unless otherwise stated. They can be generated and installed as wished on each machine.

3.2.1 Target source file

There can be more than one of such files. I have received from OAP two standard files, one for a soft (coronal) source, one for a hard (Crab-like) source and perhaps one for a diffuse source will be added.
Since the above files were generated with the wrong effective area, I have now generated locally a new reference file for a 1/100th of a Crab as described in section 5.
The intensity of the file could be changed back and forth in a reversible way modifying the arrival times with the `scaler` program. It is also necessary to shift the target off-centre with the `relocate` program. The following shifts (in micron) shall be applied for the different cases :

CCD type	Δx	Δy
EEV new focal arrang.	0	0
MAXI usual modes	3600	2000
MAXI burst mode	3600	28000

The new reference file is called `crab_after_mirror` in `/poseidon/lucio/epic/data/crab.`

In addition files generated by `linefield` (with a monochromatic line uniformly illuminating the whole FOV) are kept in files `line`*`xx`* under `data/lines`.

3.2.2 Sky file

There is a single reference sky file from OAP, incorporating QSOs, field stars and diffuse background files. Such files are also available separately and have been merged with a prototype of the `merger` program (the operation was done ad hoc since not all input files were sorted in time).

The standard reference file is called `background`, and is a merger of `qso`, `fieldstars` and `diffuse` (the latter files are in angular coordinates, but the merged file is a focal plane file). They are kept in `data/bkg`.

These files are to be considered obsolete (they refer to a wrong optics area, hence could be used as worst case overestimating the background)

3.2.3 Particle background file

There is no FOV level particle file. `cos_ray` will produce files at post-detection chip level. However since the location of the chip on the FOV has no effect on cosmic rays, it will be enough to create a file for each chip size, e.g. `cosmic_eev_d`, `cosmic_thomson_d` and `cosmic_maxi_d` (under `data/cosmic`).

The standard reference files will be created with a solar activity of 2 (intermediate).

In the case of MOS imaging mode, the reference file `cosmic_eev_d` is a merger of two files `cosmic_eev_A` and `cosmic_eev_B` (as described in section 5). For the timing mode (FW) use file `cosmic_eev_Aa` (600∞1200 pixel, image section as `cosmic_eev_A`, store section empty).

3.2.4 Derived files

There will be a set of chip files derived from each of the above target and background files for each of the standard chip configurations described below in section 4. These files will be named adding a suffix made of *`device`* (`eev`, `th` or `maxi`), *`number`* (`1`, `2` or `3`) and `_c` : e.g. `background_eev1_c`. This makes a max of 3 files for EEV, 2 for Thomson and 3 for MAXI.

For each chip file there will be a corresponding detection file, using suffix `_d` instead of `_c`.

The policy and naming convention for pre-readout file merging is yet TBD.
For each detection file there will be a corresponding readout file for each applicable mode with suffix `_r`*`mode`*, e.g. `_r` or `_rff` for full frame, `_rfs` for frame store, `_rrfs` for refreshed frame store, etc. Note that not all modes are applicable to all chip configurations (e.g. frame store is applicable to EEV chip configuration 1 but not to 2 or 3).

4 Tables of standard configurations

This section reports the standard command files used to define standard chip layouts or characteristics, and other (e.g. optics) characteristics files. The command files reported below shall be self-explanatory. In general a command file contains the replies to the questions prompted by a program (one reply with more than one value stays all on one line). It is optional to follow a reply with an exclamation mark and a comment (ignored by the program). An exclamation mark in column one means the command file will unconditionally prompt to the terminal.

The standard command files described below reside in directory `/poseidon/lucio / epic/parameter.`
The "calibration" reference files reside instead in `$XASTOP/calib/xmm/epic1` or `epic3` (whenever they are equal for MOS and MAXI, a link is made to a single version).

4.1 Mirror characteristics

4.1.1 Focal length

A focal length of 7500 mm is currently used as argument to the `focalplane` command (and is assumed via a `PARAMETER` in `randxy` when converting arcsec to micron).

4.1.2 Geometric area

A "calibration" reference file containing a nominal mirror geometric area is `geometric.area`. The value there is arbitrary and currently set to 3000 cm^2.

4.1.3 Mirror transmission

A "calibration" reference file with the on-axis mirror transmission is `mirror.transmission`. This file is generated dividing an effective area file `mirror.area` by the content of `geometric.area`. This separation occurs only for convenience of the `randomize` program (which hooks to a geometric area), so that one can use spectral normalizations in photons/cm^2/s/keV.

The `mirror.area` file is a copy of a column in a set of files (containing also the effective area for other off-axis angles) supplied by B.Aschenbach to C.Erd of SSD, which are available separately.

4.1.4 Point Spread Function

Only the on-axis PSF is currently used, assuming a radially symmetric gaussian with known sigma. The following reference parameter files contain the sigma in arcsec or microns (as used by `randxy` in "new file" or "old file" mode).

file `newpsf.command`

```
!                                       ! output file (created by RANDXY)
10000                                   ! number of events
12.72                                   ! PSF sigmas in arcsec
923456781                               ! random integer (large integer)
Simulated_point_source                  ! object description
```

file `oldpsf.command`

```
!                                       ! output file (created by RANDXY)
462.512                                 ! PSF sigmas in micron
923456781                               ! random integer (large integer)
```

4.2 Filter configurations

Filter configurations are not described by command files, but by ASCII tables with the filter transmission versus energy (located in `$XASTOP/calib/xmm/epic1` or `epic3`). At the moment there are no filter transmission tables, pending the definition of the filters.

Similar transmission files shall also be present for the silicon (or silicon dioxide) dead layer, taking into account also any geometric partial covering factor. At the moment there are no such files awaiting definition of an equivalent thickness (kg/m^2) of material by CCD groups.

4.3 Chip configurations

4.3.1 Real configurations

These configuration files correspond to representative positions in the focal plane arrangement. In general the number of configurations is kept small taking advantage of symmetries. It is however relevant to be able to handle separately the outer chips which may fall only partially within the field of view. See enclosed figures.

4.3.1.1 EEV

The new reference files refer to the latest 7-chip 2-tier focal plane arrangement. A maximal inter-chip gap of 400 µm has been used (but for chip 2 and 1 since they are on different levels, for which a zero-gap has been used).

file `eev_chip1a` describes the image section of the EEV central chip.
file `eev_chip1b` describes the store section of the EEV central chip (this file explicitly specifies different x and y pixel sizes). *However it does not describe the true store section on the right hand side, but a dummy one located on top of chip 1.*

eev_chip1a

```
!                              ! input file (full FOV in mm)
!                              ! output file (chip in pixel)
40                             ! pixel size (micron)
600 600                        ! chip size (pixels x and y)
0 0                            ! chip centre location in micron (central chip)
MOS_EEV_central_image_sect_A   ! identifier
0.1                            ! low energy cutoff
```

eev_chip1b

```
!                              ! input file (full FOV in mm)
!                              ! output file (chip in pixel)
40 12                          ! pixel size (micron)
600 600                        ! chip size (pixels x and y)
0 15600                       ! chip centre location in micron (central chip)
MOS_EEV_central_store_sect_B  ! identifier (this is a dummy sect B on TOP of sect A)
0.1                             ! low energy cutoff
```

4.3.1.2 Thomson

This section has been removed.

4.3.1.3 MAXI

`maxi_chip1`

```
!                              ! input file (full FOV in mm)
!                              ! output file (chip in pixel)
150                            ! pixel size (micron)
64 200                         ! chip size (pixels x and y)
4800 15000                     ! chip centre location in micron (centre-top)
MAXI_centre_top                ! identifier  (Y=-15000 for centre-bottom)
0.1                            ! low energy cutoff
```

maxi_chip2

```
!                              ! input file (full FOV in mm)
!                              ! output file (chip in pixel)
150                            ! pixel size (micron)
64 200                         ! chip size (pixels x and y)
14400 15000                    ! chip centre location in micron (middleright-top)
MAXI_middle_right_top          ! identifier  (Y=-15000 for middleright-bottom)
0.1                            ! low energy cutoff
```

maxi_chip3

```
!                              ! input file (full FOV in mm)
!                              ! output file (chip in pixel)
150                            ! pixel size (micron)
64 200                         ! chip size (pixels x and y)
24000 15000                    ! chip centre location in micron (outer right-top)
MAXI_outer_right_top           ! identifier  (Y=-15000 for outer right-bottom)
0.1                            ! low energy cutoff
```

files maxi_chip1, maxi_chip2 and maxi_chip3 describe the three chips in the first quadrant, from the one close to the centre, to the intermediate one, to the outer one. The remaining 9 chips can be obtained by symmetry. Perfect butting (no gaps) is assumed.

4.3.2 Auxiliary configurations

The files called eev_centre, thomson_centre and maxi_centre correspond to a chip centered in the field of view. This has no correspondence in the focal plane arrangement, but could be useful in tests in which the target is located in the origin.

4.4 Chip physical characteristics configurations

4.4.1 EEV

file eev is now a link to eev_bulk ; a variant eev_epitaxial is provided; both according to conversation with A.Holland Jan 94.

```
!                                      ! input  file (from CHIPSIZE)
!                                      ! output file
923456781                              ! random integer (large integer)
.00368                                 ! energy (keV) per hole/el. pair
.115                                   ! silicon Fano factor
front                                  ! CCD illuminated side (back or front)
70                                     ! depletion depth (microns)
30                                     ! field free depth (microns)
0 bulk 30 epi                          ! substrate depth (microns)
10.                                    ! substrate diffusion lenght (microns)
0. bulk 1 epi                          ! reflectivity of the field free boundary
2.0E12                                 ! impurity concentration (cm-3)
173.                                   ! CCD temperature (∞K)
2                                       ! 2 half side of the square (pixels)analyzed
for charge splitting
```

4.4.2 Thomson

This section has been removed.

4.4.3 MAXI

file `maxi`

```
!                                      ! input  file (from CHIPSIZE)
!                                      ! output file
923456781                              ! random integer (large integer)
.00368                                 ! energy (keV) per hole/el. pair
.115                                   ! silicon Fano factor
back                                   ! CCD illuminated side (back or front)
270.                                   ! depletion depth (microns)
0.                                     ! field free depth (microns)
0                                      ! substrate depth (microns) ignore epitaxial
10.                                    ! substrate diffusion lenght (microns) n/a
0.                                     ! reflectivity of the field free boundary
2.E12                                  ! impurity concentration (cm-3)
183.                                   ! CCD temperature (∞K)
2                                      ! 2 half side of the square (pixels)
                                         analyzed for charge splitting
```

4.4.4 Cosmic ray command files

The standard command files for `cos_ray` will be called `eev_cosmic_a`, `eev_cosmic_b` and `maxi_cosmic`, and will be generated using `precosmic` from files `eev` and `eev_chip1a` or `eev_chip1b`, `maxi` and `maxi_chip1`.

4.5 Operating mode readout configurations

4.5.1 Full frame mode

Note that the number of parameters to be supplied for MAXI is different (there is no choice of the number of output nodes, and the significance of the major and minor clock tics is different).

4.5.1.1 EEV

This section has been removed, since the mode is no longer in use.

4.5.1.2 Thomson

This section has been removed.

4.5.1.3 MAXI

file `maxi_ff`

```
!               ! infile
!               ! outfile
                ! no of frames = ALL
20              ! major tic (microsec)
                ! integration time (sec) = computed default = 44 ms
```

4.5.2 Frame store mode

there are two files `eev_fs` and `eev_fs2`, both are to be used with the `wfs_dumb` program (on Unix a script `fs_dumb` acts as a front end and allows optionally to use the "older" fs_dumb program), since FS mode is now a case of the Window mode where the window covers the entire chip image area.

Note also that for compatibility with the older version of the program (which had just two kind of clock tics) the current version of the MOS program require to specify the "major tic" as a *couple* of values. The major tics are the row shift (2 µs) and the pixel sampling (5 µs), while the minor tic is the pixel shift (1 µs).

file `eev_fs`

```
!               ! infile
!               ! outfile
                ! no of frames = ALL
2,5             ! major tic (microsec)
1               ! minor tic (microsec)
1               ! number of nodes
600 600         ! window size
0 0             ! window lower left corner
                ! integration time (sec, KEEP DEFAULT)
```

file `eev_fs2` is identical but allows readout from 2 nodes

4.5.3 Refreshed frame store mode

file eev_rfs

```
!               ! infile
!               ! outfile
                ! no of frames = ALL
2,5             ! major tic (microsec)
1               ! minor tic (microsec)
2               ! nodes
                ! integration time (sec, KEEP DEFAULT) 0.060
```

4.5.4 Window mode

4.5.4.1 EEV

file eev_wfs

```
!               ! infile
!               ! outfile
                ! no of frames = ALL
2,5             ! major tic (microsec)
1               ! minor tic (microsec)
1               ! number of nodes
100 100         ! window size (81=3HEW)
250 250         ! window lower left corner
                ! integration time (sec, KEEP DEFAULT)
```

4.5.4.2 Thomson

This section has been removed.

4.5.4.3 MAXI

currently defined only for the small window mode in file maxi_w

```
!               ! infile
!               ! outfile
                ! no of frames = ALL
20              ! major tic (microsec)
1               ! minor tic (microsec)
48 30           ! window size (NOTE y=30 not 20, since PSF is bigger)
0 0             ! window lower left corner
```

4.5.5 Fast window mode

(now called timing mode) for historical reasons in file : eev_fw

```
!               ! infile
!               ! outfile
                ! no of frames = ALL
2,5             ! major tic (microsec)
1               ! minor tic (microsec)
54,54           ! window size
273             ! window x-column origin
```

4.5.6 Timing mode

4.5.6.1 EEV

This section has been removed (it did refer to the old timing mode; the new timing mode is described above as Fast Window mode)

```
!              ! infile
!              ! outfile
               ! no of cycles = ALL
10             ! major tic (microsec)
1              ! minor tic (microsec)
FS             ! operates on frame store chip
```

4.5.6.2 Thomson

This section has been removed.

4.5.6.3 MAXI

file maxi_t

```
!              ! infile
!              ! outfile
               ! no of cycles = ALL
20             ! major tic (microsec)
1              ! minor tic (microsec)
```

4.5.7 Burst mode

file maxi_b is identical to file maxi_t (read "*frames*" for "*cycles*")

4.6 Pattern libraries

Pattern library files reside in the appropriate `$XASTOP/calib` subdirectory, together with filter transmission files *et sim.* There are two types of files, raw and compiled.

Raw files, of type `patternlib`, are created very easily in an intuitive way with an editor, and contain 32 "items" of the following form :

```
9              the pattern number (1 to 32, in sequence), followed by 5 data lines
.---.
.-X--
.-XX-
.----
.....
```

defining the content of the pixels in the 5x5 matrix (exactly as "copied" from a figure, the top line is the first one in the file). An `X` indicates "full" pixels which are part of the pattern. An hyphen `-` indicates pixels in the guard ring mask. A dot `.` indicates all remaining pixels (below threshold). Unused patterns have all empty pixels.

Compiled files, or type `patlib`, describe the same information in compact form. They are produced by program compattern, and have 32 ASCII records of the following form :

```
npat  patweigtht  maskweigth  maskbits
9     176         15551424    0000001111010010101101110
```

The pattern number is followd by a "weighted" description, by which all "full" pixels in the central 3x3 matrix are assigned one of the following 9-bit binary weights. A similar 25-bit weigth describes the mask (but is unused) while a 25 "bit array" is the one actually used to describe the mask (1 values correspond to hyphens, and 0 to everything else, and data follow in the `((x=1,5),y=1,5)` order).

```
.    .    .    .    .
.    64   128  256  .
.    8    16   32   .
.    1    2    4    .
.    .    .    .    .
```

The standard pattern library files are called `pag15` and `fig11`, respectively referring to the quoted pages or figures of the Saclay EDU architecture document (reference [4]). The one to be used currently is `pag15`.

4.7 Auxiliary command files

Auxiliary command files can be used for convenience to generate images or spectra with programs `xgaccum` or `xhaccum`. They are not described in detail here, but just mentioned. By "chip file or alike" it is intended an `_c`, `_d` or `_r` file with the only condition that coordinates are given in CCD pixels.

`arcsecimage`	to generate an image from a file in angular coordinates
`fov_mmimage`	to generate an image from a focal plane file
`eevimage`	to generate an image from an EEV chip file or alike
`maxiimage`	to generate an image from a MAXI chip file or alike
`allhisto`	to generate a charge spectrum from any _d photon file

5 Usage examples

This section presents some real cases of using sequences of EPOS commands.

5.1 Generating a 1/100th Crab photon file

`cd ~/epic/parameter`	Go where command files are

`xasset spacecraft xmm`
`xasset instrument epic`
`xasset rootdir ~/epic`
`xasset datadir data`
	Set up the environment ...

`xasset order ct`
`xasset target crab`
	Data files go in ~/epic/data/crab

`randomize powerlaw  9.8e-2 2.1 3e21 100 0.1 10 count`	Tells you there are 27171 counts before mirror

`xasset command newpsf`
`randxy crab_before_mirror 27171`
	Photon file requires 27171 events in a gaussian PSF

`randomize powerlaw  9.8e-2 2.1 3e21 100 0.1 10 oldxas crab_before_mirror 1000 123456789`	Fill energies with Crablike spectrum in photon file
`randomize sinusoid 271.71 0. 0. 100 0 100. oldxas crab_before_mirror 1.E6 123456789`	Fill the times (uniform) in photon file
`header_edit crab_before_mirror tmax1 100000000`	
`filter crab_before_mirror crab_after_mirror mirror 987654321`	transmit thru mirror
`focalplane crab_after_mirror 7500`	convert arcsec to micron
	dead layers and filters ignored

In the case of MOS then proceed

`xasset command eev_chip1a`
`chipsize crab_after_mirror crab_eev1_c`
	convert mm to pixel in central chip image area for MOS

`header_edit crab_eev1_c chipsize 600 1200`	fudge the chipsize to include store area

`xasset command eev`
`ccd_det crab_eev1_c crab_eev1_d`
	simulate the detection

In the case of pn instead proceed

`relocate crab_after_mirror 3600 2000`	offset target from chip centre

`xasset command  maxi_chip1`
`chipsize crab_after_mirror crab_maxi1_c`
	convert micron to pixel in one pn chip

`xasset command maxi`
`ccd_det crab_maxi1_c crab_maxi1_d`
	simulate the detection

5.2 Generating a cosmic-ray file

The example shown is for the MOS case, imaging mode

`precosmic eev eev_chip1a eev_cosmic_a`
`xasset command eev_cosmic_a`
`cos_ray cosmic_eev_A ,, 200`
	generate a file for image section

`precosmic eev eev_chip1b eev_cosmic_b`
`xasset command eev_cosmic_b`
`cos_ray cosmic_eev_B ,, 200`
	generate a file for store section

`relocate cosmic_eev_B 0 600`	shift the y-coordinates to appear on top of the image section

`merger cosmic_eev_A cosmic_eev_B NULL`
`  cosmic_eev_d NOFLAG`
	merge the two files

`header_edit cosmic_eev_d chipsize 600 1200`
`header_edit cosmic_eev_d chip MOS`
	fake chip size and name in header

`xasset command eev_fs`
`fs_dumb  cosmic_eev_d cosmic_eev_r`
	simulate readout in FS mode

`emce_i cosmic_eev_r  cosmic_eev_e 25 2750`
`  pag15 Y`
	simulate EDU action

In the case of Timing mode (FW) ignore store section :

`precosmic eev eev_chip1a eev_cosmic_a`
`xasset command eev_cosmic_a`
`cos_ray cosmic_eev_A ,, 200`
	generate a file for image section

`copy cosmic_eev_A cosmic_eev_Aa`
`header_edit cosmic_eev_Aa chip MOS`
`header_edit cosmic_eev_Aa chipsize 600 1200`
	just fudge chip size and name

`xasset command eev_fw`
`fw_dumb cosmic_eev_Aa cosmic_eev_rfw`
	proceed with readout

`emce_t cosmic_eev_rfw cosmic_eev_efw 25`
`  2750 Y`
	and simulate EDU action

5.3 Looking for the limiting rate

Start from a post-detection file for the 1/100th Crab spectrum, and scale its intensity up and down until there is excessive pile-up. Example shown for MOS.

```
cd ~/epic/data/crab                          go where data are
copy crab_eev1_d ref                         make a working copy
cd ~/epic/parameter                          go back

scaler ref 0.01                              scale to 0.1 mCrab

startloop:
      xasset command eev_fs                  simulate readout
        fs_dumb ref prova_fs_1

        emce_i prova_fs_1 test_fs_1 25 2750  simulate EDU
        pag15 Y
                                             go into IDL and evaluate pile-up
        idl ...
        scaler ref 5                         scale to 0.5 mCrab

goto startloop:
continue scaling up or down until excessive pile-up
```

For pn there is one step more in the cycle (simulate on-ground split event reconstruction) :

```
xasset command maxi_ff
ff_dumb ref prova
epce_i prova prova_epce_ff 25 6793 6793
ground_pi prova_epce_ff test_epce_ff 25 2750 pag15 n
```

5.4 Merge source and cosmic-ray files

This would be the first step of a "real world" simulation (which should also include the x-ray background, here the first NULL takes place of the X-ray background file).

```
xasset target <CR>                           look for data files in
xasset order <CR>                            subdirectories of ~/epic/data

merger NULL crab/test cosmic/cosmic_eev_d    merge files with flagging !
  merged/prova FLAG

xasset command eev_fs                        simulate readout in FS mode on
fs_dumb prova prova_r  92                    overlapping time interval (200 s
                                             = 92 frames)

emce_i prova_r  prova_e 25 2750 pag15 Y      simulate EDU action
```

Appendix A : how to use EPOS

This appendix describes how to use EPOS programs already existing and officially installed. This section is so far relevant to the installation at IFCTR. If you want to develop your own programs refer to Appendix B.

Not all software is necessarily installed on all systems listed, although it can be loaded when needed (the Ultrix version is always available). All arrangements are subject to change without notice.

AA.1 On Vax

You might be used to invoke a program in VMS by doing `RUN program.` This will not be adequate to EPOS programs, since you cannot pass arguments to the runstring if you use `RUN`. You shall instead define a foreign symbol pointing to each program (see VMS documentation, in general you do something like `program == "$disk:[directory]program"` , and then you can just say `program` followed by any argument).

EPOS executables are in `DUA0:[LUCIO.EPIC.BIN]`; non-EPOS XAS executables (which may be useful to you) are in `DUA0:[LUCIO.XAS.BIN]`.

If you want not to be bothered by doing this for each program (and may be you are interested in having in VMS something like the Unix or MS-DOS path), you can do the following :

Include in your `LOGIN.COM` the instructions

`$@DUA0:[LOCAL.PROGTOOL]PATH DUA0:[yourself.BIN]` or wherever you keep your programs
`$@DUA0:[LOCAL.PROGTOOL]REHASH` to make programs known to the system
`$@DUA0:[LOCAL.PROGTOOL]PATH PREPEND []` to add the current directory in front

at any time you can also append to the path the EPOS and XAS directories with

```
PATH APPEND directory list
```

(or you could put the directories after your private `BIN` directory in the first line in `LOGIN.COM`). If you define a PATH manually, you may need to do a `REHASH` to make all programs known (unless you compile them yourself with `comlink`, see below).

AA.2 On Unix

In order to call EPOS programs by name, they shall be in your path. If you do `echo $path` or `printenv PATH` you will see what your current path is. The simplest way to add a new directory to the path is to do :

```
set path = ($path newdirectory)
```

Of course the new path is known only in the window in which you did it (and in whatever window you create from there). If you want to have it done all the times, you should create a file `.mypaths` in your home directory (see the example in `/poseidon/lucio`).

Mind that executables for Sun and DECstation are different, and they are located in different places. In particular :

For Sun EPOS executables are in `/home/lucio/epic/bin` ; non-EPOS XAS executables (which may be useful to you) are in `/home/lucio/xas/bin`.
For DEC EPOS executables are in `/poseidon/lucio/epic/bin`; non-EPOS XAS executables (which may be useful to you) are in `/poseidon/lucio/xas/bin`.

AA.3 Using command files

EPOS (and XAS) programs do not normally use command files, nor are parameter file driven. If you call a program like `focalplane`, it will prompt you for what it needs. If you instead do `focalplane file length` it will use the parameter you pass. If you do `focalplane file` it will prompt your for the missing parameters.

However you may write all parameters in a command file (see examples in `/poseidon/lucio/epic/parameter`), and "activate" such command file by doing `xasset command filename`. If you then call a program by name it will read from the command file instead than from the terminal.

A command file is written one line for each question asked by the program (if the answer to a question is more than one value they must be put on a single line).

In general you want to have in a command file a "standard setup" and be prompted for things like file names. You may prepare more command files (say one for EEV CCDs, one for Thomson and one for MAXI) and activate the one you want. You may specify a parameter (like a file name) with an exclamation mark (`!`) to force that parameter to be asked at the terminal.

If you pass (some) parameters in the run string (e.g. `ccd_det file1 file2`) they override the command file. You can also omit some parameters in the run string using commas to skip them : in such case they will be read from the command file (if active) or the terminal (otherwise).

The mechanism to be used to activate a command file is now working. You just do :

```
xasset command commandfile
```

before issuing any EPOS or XAS command. Note that `precosmic` will set up the command file it creates for you (no need to do `XASSET`). A command file definition will not survive the invocation of the program it is used for (unless set to permanent, ask me if interested).

There are two known problem in Unix : first of all the XAS environment is saved to files named `ttyp`*`n_machine`*`.environment` in your home directory (~), e.g. `ttyp1_poseidon` if you are running from pseudoterminal ttyp1 on machine poseidon. These files are preserved from one session to the next, and automatically reset by first command of each session. In case of occasional trouble you might need to delete them.
Also occasionally (on Ultrix) the `xasset` command terminates with a core dump. This occurs when the sum of the number of pre-existing XAS environment variables (lines in the above file) and Unix environment variables (e.g. `printenv | wc -l`) is equal to an installation dependent number (with current installation this is 31). In such case just create one or more dummy Unix variables (`setenv DUMMY`) and repeat the `xasset`.
This problem should have disappeared with XAS 1.1 since August 94.

AA.4 Setting the environment

Using the above described path mechanisms you are able to run EPOS programs from any directory. In general you run it from the directory where the data are, but you may want to arrange it otherwise. For instance I run it from the directory where the command files are, and keep the data elsewhere.

If you specify a filename including a full path, this is one way to indicate files are elsewhere. Your paths override always. EPOS and XAS program want file paths in a "Virtual Operating System" format, which is Unix-like : in fact it is identical to the Unix path in Unix, while in VMS you translate something like `DUA0:[SANDRO.EPIC]` as `/dua0/sandro/epic` or `/sandro/epic` (the disk can be omitted if it is the default disk), or a relative subdirectory `[.EPIC.DATA]` as `epic/data` etc.

In general you do not need to specify the file type (`.oap`, `.photon` , `.image` or `.calib`) but if you do you override the system default.

However there is another handier way to specify where files are. This is done setting some global variables. I assume here you have two subdirectories of a common root, called `/home/pinco/palermo` and `/home/pinco/data`, and a third directory `elsewhere`. If you do the following definitions :

```
xasset ROOTDIR /home/pinco
xasset FOTDIR palermo
xasset DATADIR data
```

and you work from elsewhere, the program will look for `.oap` files in `/home/pinco/palermo` and for all other files in `/home/pinco/data`.

For some programs (typically `filter)` it is necessary to do some other assignments. You need to do the following once :

```
xasset SPACECRAFT xmm
xasset INSTRUMENT epic
```

while the following shall be done in your `LOGIN.COM` or `.mychsrc` file :

```
VMS :   $ DEFINE $XASTOP DUA0:[LUCIO.XAS.]
Sun:    setenv XASTOP /home/lucio/xas
DEC:    setenv XASTOP /poseidon/lucio/xas
```

AA.5 General XAS utilities

The following commands are not part of EPOS but are useful :

`hlist`	to look at the content of a file header
`tlist`	to list the content of a photon list
`tofits`	to convert a XAS file to FITS
`xgaccum`	to accumulate an image
`xhaccum`	to accumulate a spectrum
`saodisp`	to know how to call SAOImage
`header_edit`	to add or edit an header keyword
etc.	

As an undocumented addition to EPOS, a command `xzaccum` allows (in a way similar to xgaccum) to accumulate a charge image (instead of a photon image) or an image having any other field on the Z-axis.
There are more of such undocumented additions now, which are specialized in a particular accumulation like `chargeaccum` or `flagaccum` (the latter accumulates images with the split, smear or pileup flags).

AA.6 Using with SAOimage, IDL or MIDAS

XAS files can be looked at and manipulated with several other packages. One can use SAOImage, IDL, MIDAS or IRAF for images, and IDL or MIDAS and STSDAS for any other file (which appears as a table).

To display a XAS image with SAOimage, call `saodisp`, and then copy (better, cut-and-paste) the command which `saodisp` writes on the screen. You can even do the following : `` eval `saodisp imagename` `` (the quotes are *back*-quotes : if you know Unix you know what it means ! Note however that using `eval` interferes with the history recall of csh).

To read any XAS file within IDL, a procedure is available, which can be called as `xasread,`*`file`*`,`*`structure`* (e.g. `xasread,'pinco.image',myima`). The named structure will have several components, which can be viewed with `help,/stru`. One set of components (*`structure.keyname`*, e.g. `myima.naxis1`) correspond to XAS header keywords. Another component is called `structure.data`, and is respectively a 2-d array (`myima.data`) in case of images, or another structure of variable layout (do `help,/stru,`*`structure`*`.data` to see its component, which are usually table columns of self-explanatory names, e.g. `myphot.data.x`, `myphot.data.charge`).
The data (sub)structures can be displayed and operated with usual IDL commands.
A set of pseudo man-pages for the IDL XAS interface is available c/o the author.

To move a file to MIDAS or IRAF, use `tofits` to convert it to FITS and handle it as normal FITS file. In IRAF table files must be read with STSDAS `strfits` command.

Appendix B : how to develop EPOS programs

This section is dedicated to those who want to develop programs for EPOS, either privately, or to be later inserted in the official suite (mantained by the author).

Please note that I currently use my own set of utilities (the *Real Programmer Tool*, documented separately, see reference [5]) to handle, compile and link all programs in an uniform way on all systems. If you want to use something else (`make`, `.COM` files or whatever) it is you own business only.
Contact the author to make sure your account is set up to use the Real Programmer Tool.

Also note that a re-organization of the XAS libraries took place in Aug 94, and anyhow the arrangements of XAS software are **subject to change without notice**. For the latest semi-official release of XAS libraries see the installation info at `http://botes2.tesre.bo.cnr.it/Xas/` : this is up-to-date at the so-called XAS 1.1 version of Aug 94. One of the HTML pages there contains a script which can be used to see which modules have been changed since then both in the master version and in my private development version (which is the one referred to in the present note, i.e. the one on `/poseidon/lucio/xas`). Note that I mantain regularly the development version on Ultrix, and only occasionally update the Vax and Sun ones.

When developing software you need to decide only three things : where do you keep source files, where do you keep executables, where do you keep temporary (object) files. You will not be developing library subroutines, so you can use the default XAS places.

I would strongly encourage that you use the same sources for VMS and Unix : by this I mean two things. One is that you **must** write code which is **not VMS dependent** (do **not** use tabs, do **not** use inline comments with !, do **not** use `DO-ENDDO`, etc. : **contact me** for a check). The second one is that you locate the sources on a Vax directory and make them accessible from Unix; to do this you should do two things again : the source file shall be called `*program*.f` and not `*program*.for` ; the source file shall be in `STREAM_LF` format (for this, just after you created it do `streamlf *filename*`, and your file will be converted once forever).

I will now assume that your sources are in `[YOU.EPIC.SOURCE]`, that you want to put VMS executables in `[YOU.EPIC.BIN]` and Sun executables in `/home/you/epic/bin`, and that your temporary area is `[YOU.TEMP]` and `/home/you/temp` respectively.

Now go in some place (for instance `[YOU.EPIC]` and alike) and create a file called `configure` (on Unix) or `CONFIGURE.COM` (the name does not matter) containing :

Unix	VMS
sourcedir /vax/you/epic/source	$sourcedir [you.epic.source]
targetdir /home/you/epic/bin	$targetdir [you.epic.bin]
relocdir /home/you/temp	$relocdir [you.temp]
libsourcedir /home/lucio/epic	$libsourcedir [lucio.epic]
libdir /home/lucio/xas/lib	$libdir [lucio.xas.lib]
incdir /home/lucio/xas/include	$incdir [lucio.xas.include]

Once in a session go there and do `source configure` (Unix) or `@configure` (VMS).

To compile a link a program, deleting the relocatable and leaving only the executable, you do it all with a simple command like :

```
comlink program default option
```

`default` make references to a file `default.loader` which shall exist in your sourcedir directory, and shall contain the list of libraries you use. Copy it from myself, or type it : it is a single line with the following list, verbatim :

```
teukolsky xaslib vos general
```

`option` is instead either `list` or `term` or nothing. If you want to see the error messages, use at least `term`. `term` will work everywhere. `list` is used to look at the program listing (which is then deleted). It works on VMS (you are put in `EDIT`, quit to exit), or on a workstation (not on an Unix terminal), where the listing appears in a separate window.

If you need other libraries you can set up a different loader file. At the moment there are two other loader files, `cern` (used by `cos_ray` with the CERN libraries) and `epic` (used by the dumb readout programs to share the pileup classification routine in `epiclib`). These loader files assume the relocatable libraries are under `xas/lib`. If you have your private libraries, you might introduce paths in your loader files, but the price will be that you will need separate loader files for VMS and Unix.

To be able to use your program, you shall make sure that your targetdir is in the path. If you are developing new programs, you can just append the directory at the end of the path, but if you want to write a variant of an official program, you must make sure that your directory comes *before* the official one.

See the existing EPOS sources for examples of how XAS files have to be handled, and how the XAS user interface is called. Additional documentation is available c/o myself. Introductory concepts are avaialable at `http://www.ifctr.mi.cnr.it/ Sax/xasblurb.html`