emchain

August 16, 2005

Abstract

Generate the EPIC-MOS event list product.

1 Instruments/Modes

Instrument	Mode
EPIC MOS	IMAGING, TIMING

2 Use

pipeline processing	no
interactive analysis	yes

3 Description

The **emchain** script chains and loops over all first-level EPIC MOS tasks to produce event lists for all exposures, ready to be exported as PPS products.

3.1 Generalities

By default all events flagged for rejection during the chain are physically removed from the output file (to save space) except those `OUT_OF_FOV` (useful for cosmic-ray background subtraction) and `REJECTED_BY_GATTI` (used for proton flare rejection). To keep all events in output set `rejectbadevents`=N. To use a different mask on `FLAG` set `rejectionflag` to another hexadecimal value (binary flags are described in **evatt**, and in the header of the `EVENTS` table of the event lists).

For later imaging analysis selecting with #XMMEA_EM is advised. For accurate spectral analysis, selecting with #XMMEA_SM is advised. This removes all flagged events except those flagged only as CLOSE_TO_DEADPIX. Using the stricter selection '`FLAG` == 0' is in theory even better in terms of spectral reliability, but in practise it makes little difference and can reject a sizable fraction of events (several entire columns per CCD are declared as DEAD).

All text output is sent to standard error, and may be redirected to a file (very useful to investigate problems). It is good practice to look at the warnings sent by the tasks and **emchain** itself. If the SAS_VERBOSITY environment variable is set to 1, **emchain** will tell which tasks it called. If SAS_VERBOSITY is set to 4 (suggested value), its constituent tasks will provide useful information. Larger values of SAS_VERBOSITY may result in large output files. If SAS_VERBOSITY=0, **emchain** will be mute, except for warnings and error messages.

All files are recognized by their name. Input files are looked for in the directory entered via the `odf` parameter or the SAS_ODF environment variable, which must also contain the general ODF files (attitude, time, summary file). Output files are created in the current directory. Intermediate files are removed at the end, unless `keepintermediate`=Y or an error occurred. Intermediate and output files are overwritten unless `clobber`=Y.

If one constituent task ends in error, **emchain** will continue anyway with the next CCD, exposure or instrument unless `stoponerror`=Y.

3.2 Main loop

The main subroutine (processOdf, Fig 1) loops over all exposures and instruments (MOS1/MOS2) present in the input directory (looking for event list files).

If `withatthkgen` is true or if the tracking history file does not exist already, **atthkgen** is run first. **tabgtigen** is run on the output to generate the attitude GTI. The tolerance on attitude variations may be modified via `atttol`. **hkgtigen** is run if the HK GTI does not exist already. Those GTI files are not applied by default, they are generated for information. They can be applied by setting `filteratt`=Y and/or `filterhk`=Y. They are then merged with the user GTI (if `ingtiset` is set) into the external GTI used by **emframes**.

processOdf creates one (or two, if a CCD is operated in TIMING mode) event list for every selected exposure, from all relevant ODF material and (if they exist) the good time intervals generated by **tabgtigen** and the list of bad pixels (from the CCF or produced internally).

In a first step it loops over all CCD/nodes, calling in sequence, as shown in Fig 2:

1. **emframes** on the auxiliary file, the events file and the external GTI file (if any), creating a frame file as expected by **emevents** and a CCD/node specific GTI file which will be reinjected in the final call to **evselect**.
2. **badpix** on the events list, adding the `BADPIX` extension. If a bad pixels file exists, it is used instead of the CAL calls for the non-uplinked bad pixels, (*i.e.* **badpix** is called with `getuplnkbadpix`=Y `getotherbadpix`=N `getnewbadpix`=Y).
3. **emevents** on the events list, the offset/variance file and the frame file, creating a new events list which will be propagated through **attcalc** and **emenergy** to **evlistcomb**.
4. **gtialign** on the external GTI file and the events file, then **gtimerge** to merge the resulting aligned GTI and the CCD/node specific GTI.
5. **attcalc** on the new events list, filling the `X/Y` columns.
6. **emenergy** on the new events list, filling the `FLAG`, `PHA` and `PI` columns.

By default position, energy and time of each photon are randomised within their respective bins (one CCD pixel for position, one ADU for energy, one frame for time).

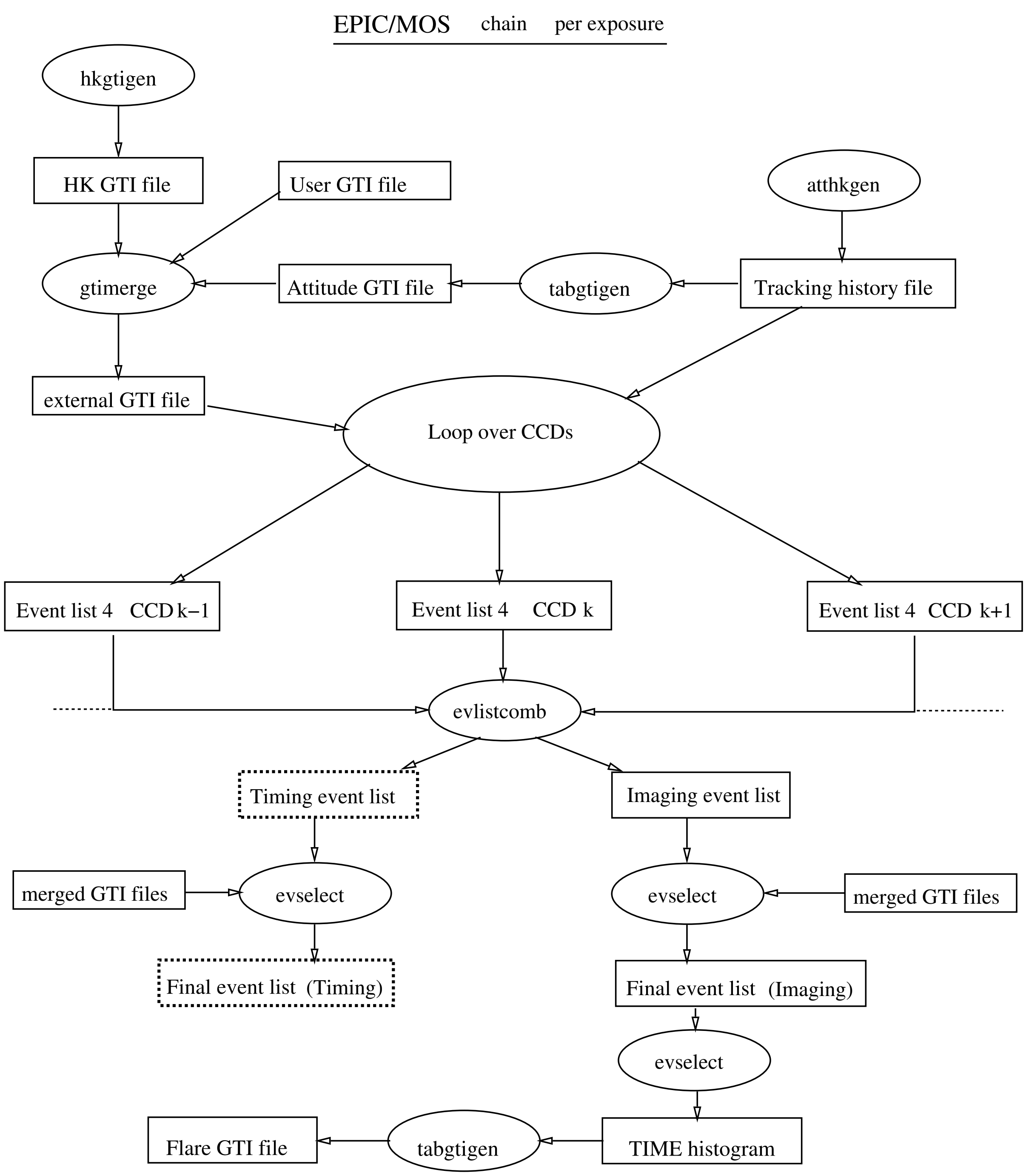

Figure 1: Organisation of the EPIC-MOS chain: merging the event lists. The files in boldly dashed boxes are used (or produced) if they exist. The files in simply dashed boxes are options of the individual tasks not used in the current chain.

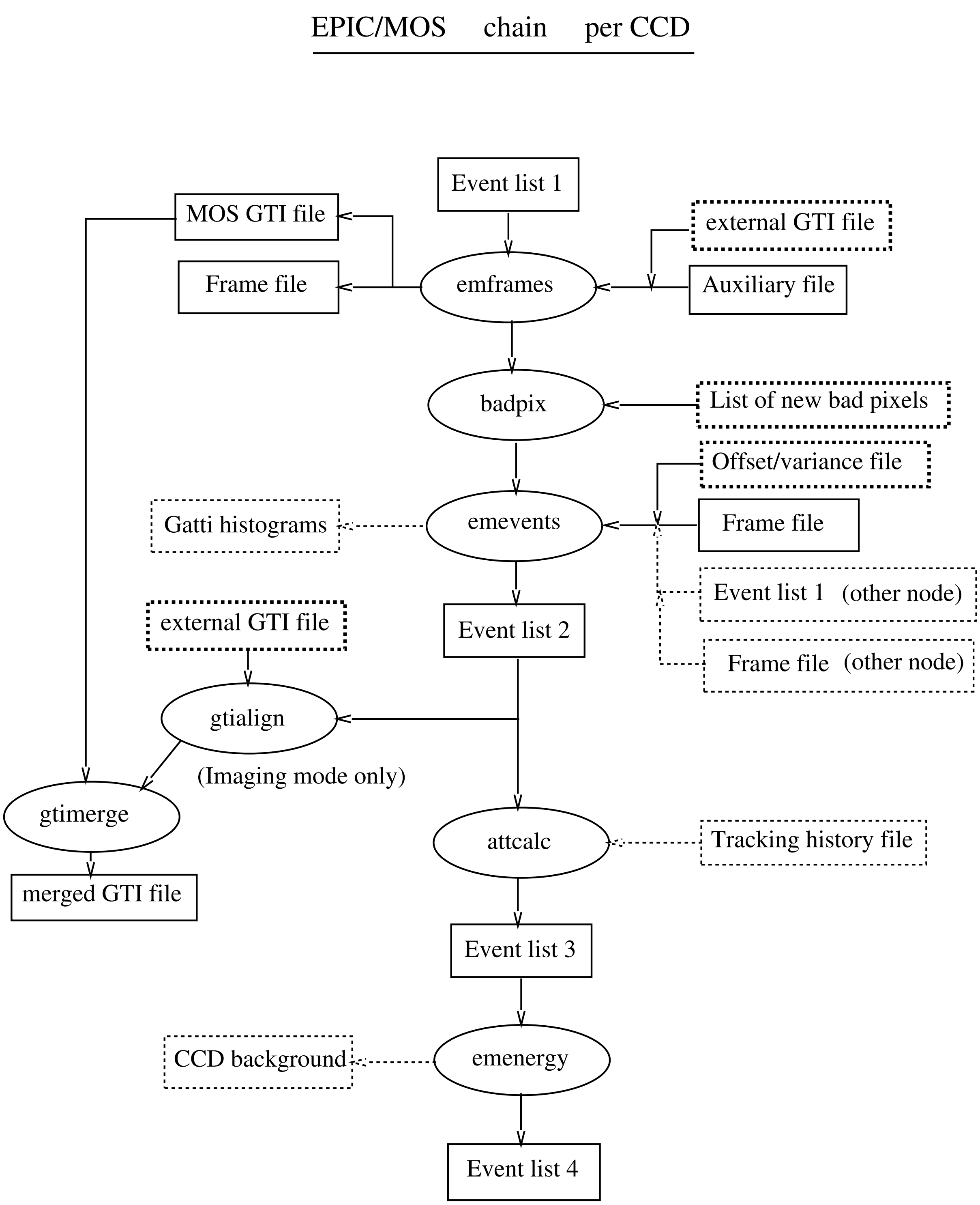

Figure 2: Organisation of the EPIC-MOS chain at CCD/node level with file inputs. Same conventions as in Fig 1

Then (Fig 1) all the event list files created (one per CCD/node) are merged by **evlistcomb**, creating one events list per mode (IMAGING, TIMING). Finally **evselect** is called on the resulting events list(s), with (CCDNR==$node$ccd) && GTI(merged GTI file,TIME) for all CCD/nodes. The list of calibration files used to analyse the data is added to the output files as a `CALINDEX` extension.

3.3 Flare rejection

If `makeflaregti`=Y, for all merged files (*MIEVLI*), an additional call to **evselect** (Fig 1) produces a light curve of single events from CCDs in Imaging mode flagged with `REJECTED_BY_GATTI` (with actual energy above 14 keV), with time bins defined by `flaretimebin` (if that time bin is too small to give reasonable statistics per bin, it will be increased automatically). This is aimed at finding efficiently periods of flaring activity when the focal plane is illuminated by low-energy protons. Those are mostly singles and are not cut-off by the mirror efficiency above 10 keV (contrary to astronomical sources). The result is divided by the area of the external CCDs within the field of view, and expressed in cts/ks/arcmin^2. A fractional exposure column `FRACEXP` accounts for incomplete time bins.

If `globalflare`=Y (default), timeseries are built for all MOS event lists in the current directory, then summed together (the `ERROR` columns are summed quadratically) to build a single global timeseries. This allows to apply the same GTI to both MOS instruments, and to improve the statistics in the timeseries.

This light curve is sent to **tabgtigen** which selects all intervals when the number of cts/ks/arcmin^2 is lower than `flaremaxrate`, and produces a Good Time Intervals file. This file is filtered to avoid intervals only one `flaretimebin` long, which are usually negative fluctuations during a moderately strong flare. This file is NOT applied to the events list, unless `applyflaregti`=Y.

The quiet level is around 0.8 cts/ks/arcmin^2. The default value of `flaremaxrate` is conservative (to avoid rejecting too many intervals) and should be all right in most situations. The default value of `flaretimebin` is such that it covers an integer number of frames (20). It should not be chosen too small if `TIME` is not randomised (no 'T' option in `randomize`).

If `applyflaregti`=Y, the resulting Good Time Intervals are applied only if they total more than 10% of the exposure time. Otherwise the events list is left unscreened. This applies in particular to the bad pixels detection (Sect. 3.4).

3.4 Bad pixels detection

In most cases relying on the bad pixels registered in the CCF is not enough. This is because the CCF stores only the bright pixels of relatively high occurrence, but bright pixels at a low level may be a nuisance as well.
If `badpixfindalgo`=EM (default) or EP, the whole analysis of one exposure (Fig 1) is run first with **emevents** and **emenergy** in a simplified mode, and skipping **badpix** and **attcalc**. Then one of two bad pixel finding algorithms is called (Fig 3), depending on `badpixfindalgo`.

- If `badpixfindalgo` is set to 'EP', **badpixfind** is called. This is a conservative algorithm which will find clear-cut bright pixels or columns.
- If `badpixfindalgo` is not set or set to 'EM', **embadpixfind** is called. This is a more sensitive algorithm which will detect bad pixels, segments of columns or rows down to the statistical limit of Poisson counts.

If part of the exposure is affected by flares, this can seriously reduce the power of the bad pixels search (flares act as noise for the bad pixels and make detecting them more difficult). Therefore an intermediate

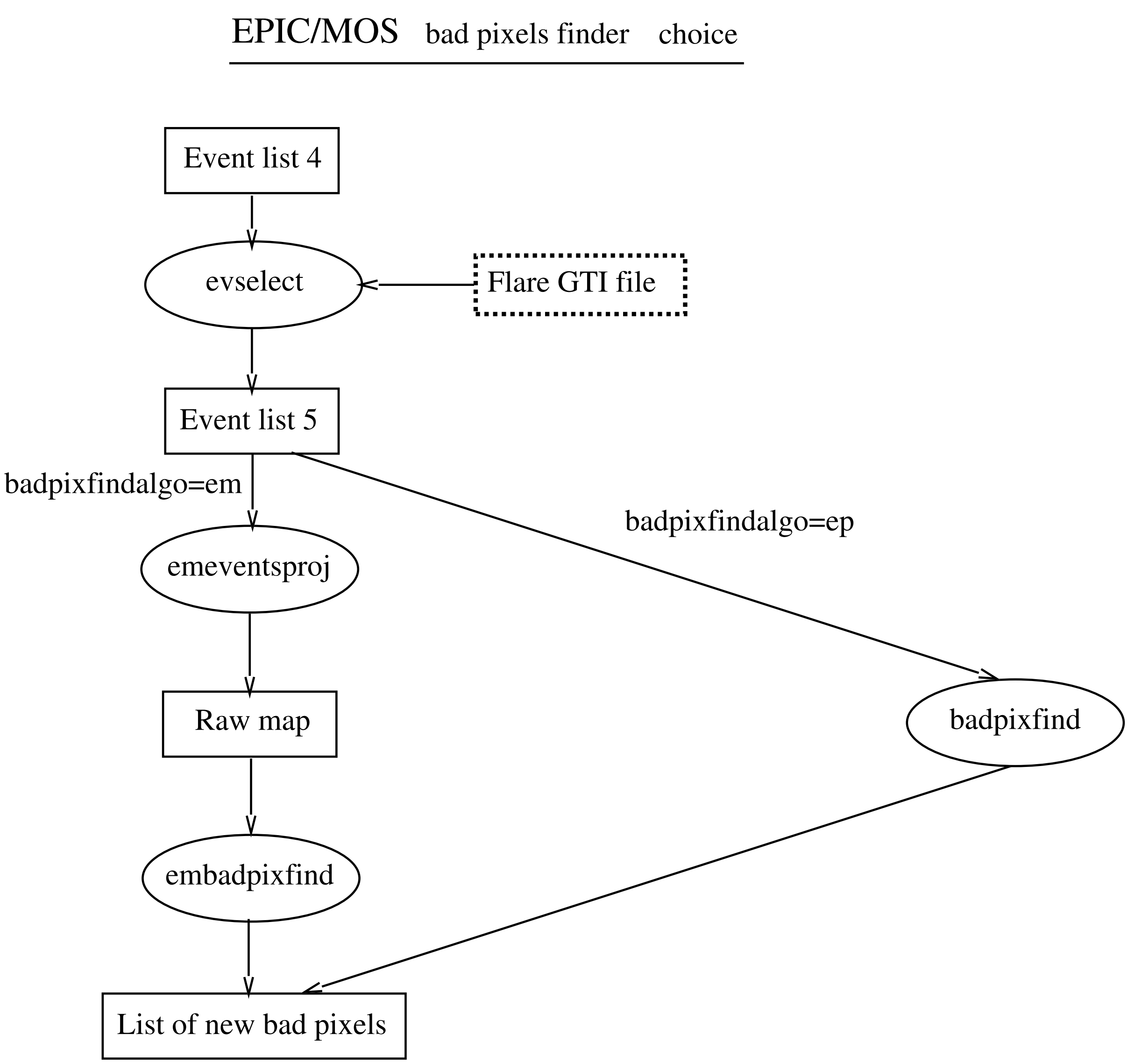

Figure 3: Organisation of the EPIC-MOS chain: Bad pixels search. Same conventions as in Fig 1

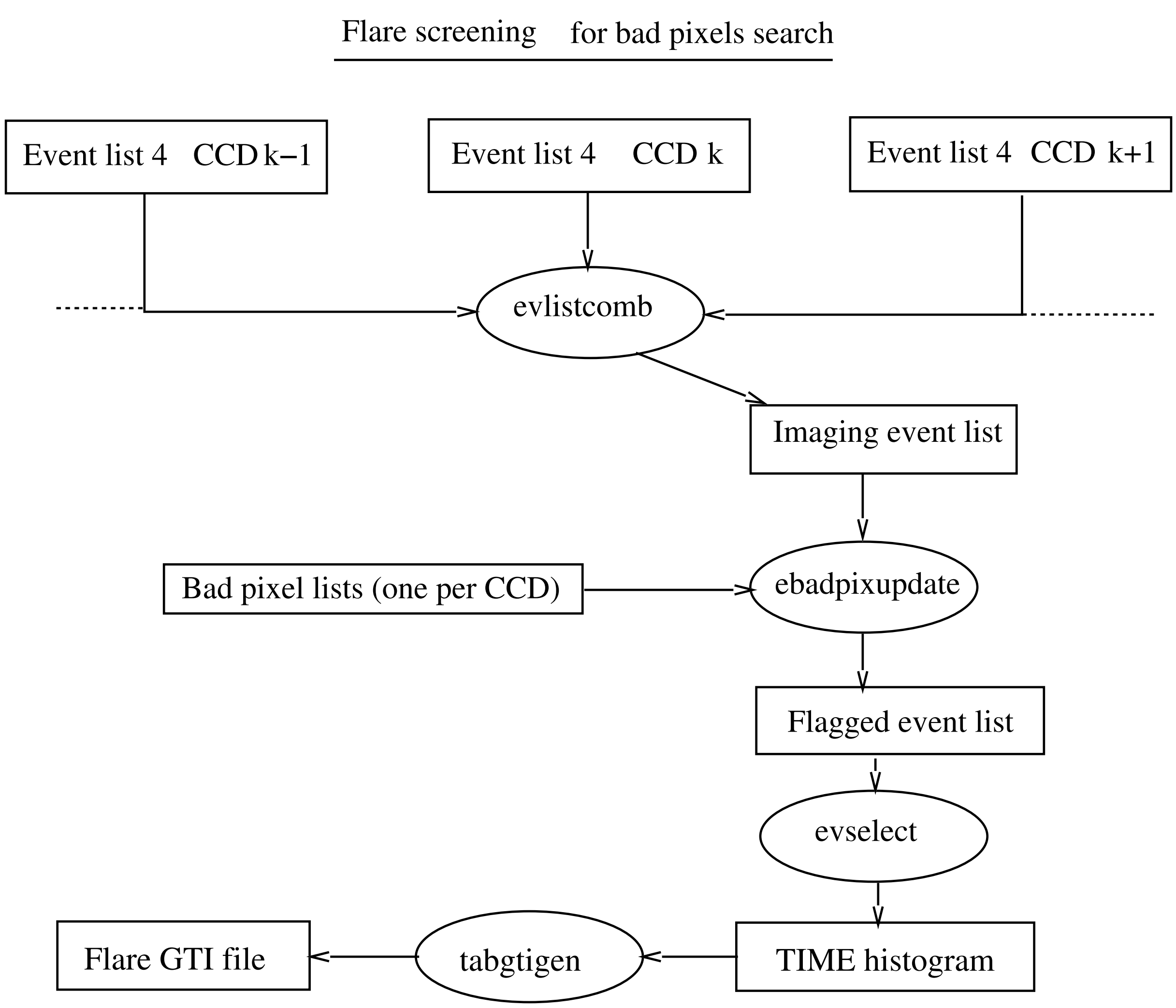

Figure 4: Organisation of the EPIC-MOS chain: Flare screening between first and second bad pixels searches. Same conventions as in Fig 1

flare screening (like in Sect. 3.3) is necessary (Fig 4). The bright pixels (which can perturb the flare screening) are flagged using **ebadpixupdate**. The resulting files are used to generate Good Time Intervals outside flares. Those Good Time Intervals are exposure specific whatever the value of `globalflare`. Then the bad pixel search is run a second time on the data outside flares, in incremental mode.

For `badpixfindalgo`=EM (or not set), the algorithm is called a third time (incrementally) on energies below 500 eV (and after flare screening), unless `lowenerbadpix`=N. This sometimes detects bad pixels more easily, because most appear at low energy.

The resulting bad pixels file is then used by **badpix** in the main pass on the data (Fig 1). Bad pixels declared in the CCF are read as well, and merged with those found in the local search. This second pass restarts just after **emframes**. The `stopafterbadpixfind` parameter allows to stop **emchain** before the main pass, to investigate in detail how the bad pixels detection worked.

If `badpixfindalgo` is set to NO, then the first loop is not done and the bright pixels are read from the CCF.

3.5 Customisation

Many options allow to run **emchain** on a subset of the data, restart from previous output, change the options of constituent tasks.

- The `exposures`, `instruments` and `ccds` parameters allow to run **emchain** on a subset of the data files (select an explicit list of exposures, instruments or CCDs).
- If `addvigweight`=Y, then **evigweight** is called to add a `WEIGHT` column to the events list (for extended source analysis).
- All individual steps in emchain except **emframes** may be skipped (`withbadpix`, `withemevents`, `withattcalc`, `withemenergy`, `runevlistcomb`, `applyccdgti`, `makeflaregti`). The whole loop over CCDs may be skipped if `runccdloop`=N (the only available functionalities are then those of Fig 1).
- If `randomize` does not contain the letter P, **emevents** is run with `randomizeposition`=N. If `randomize` does not contain the letter T, **emevents** is run with `randomizetime`=N. If `randomize` does not contain the letter E, **emenergy** is run with `randomizeenergy`=N.
- Parameters may be passed to constituent tasks. The syntax is 'taskname:paramname=soandso', where 'paramname' is the parameter name for task 'taskname'. This will be passed as 'paramname=soandso' when calling 'taskname'.
  For example, 'emenergy:useccfdarkframe=Y' allows using the CCF dark frame for the CCD background correction (this corrects for variations at the pixel scale, whereas the E_4 data allows to correct for variations on scales larger than 5 pixels).
  This syntax does not allow to specify different parameter values on different occurrences of 'taskname' within **emchain**. The additional parameter will be passed on all occasions.
  Parameters specified in that way supersede possible settings of the same parameters by **emchain** itself.
- For calibration purposes (offsets' evolution), or checking optical loading, it can be useful to look at the background (base level) of a CCD. If `writeccdbackground`=Y, then emchain writes one file per CCD and per exposure, containing the CCD background (as determined by **emenergy**). This is similar to passing `backgroundset` to **emenergy**, but allows to save all such files under a different name.

- **emchain** normally ignores data obtained with CCD read-out gain set to low (/10). This behaviour may be overridden by setting `processlowgain`=Y. Low gain data is then treated as if it was normal data, and included in the merged events file in output, in addition to normal data. Its `PI` will still be wrong, and the flare screening mechanism will not work normally (because there are very few events at energy more than 100 keV even during flares).
- If `fulloutput`=Y, all columns originally present in the events files (in the ODF) are preserved in the output file (this about doubles the size of the file).
- If intermediate files from a previous call to **emchain** still exist (for example `eventnn.in` files), **emchain** may use them as a starting point instead of the ODF (`startfromodf`=N).

3.6 Examples

emchain takes some time to run. It is usually better run as a batch job (at command). Here are a few examples of how to use it. They assume that the SAS_CCF environment variable was set to the relevant Calibration Index File.

- Standard run (get calibrated event files from an ODF located in your_odf_dir), sending the output to a log file:

```
emchain odf=your_odf_dir 2> emchain.log  (sh shell)
emchain odf=your_odf_dir >& emchain.log  (csh shell)
```

- Same, but getting the bad pixels from the CCF instead of the data and applying attitude and HK GTIs:

```
emchain odf=your_odf_dir badpixfindalgo=NO filteratt=Y filterhk=Y
```

- Select particular exposure, instrument and CCDs, keep intermediate files and stop at first error:

```
emchain instruments=M2 exposures=S002 ccds='1 3 4' stoponerror=Y  \
        keepintermediate=Y
```

- Keep events flagged for rejection and all original columns in the ODF, apply user GTI and no `TIME` randomization:

```
emchain rejectbadevents=N fulloutput=Y randomize='PE' ingtiset=hkgti.ds
```

- Rebuild the flare GTI files with different settings, keep them exposure specific and add the `WEIGHT` column:

```
emchain runccdloop=N runevlistcomb=N addvigweight=Y  \
        globalflare=N flaretimebin=104 flaremaxrate=1.2
```

- Run **attcalc** with fixed attitude:

```
emchain attcalc:attitudelabel=fixed attcalc:fixedra=204.6877  \
        attcalc:fixeddec=-27.6984 attcalc:fixedposangle=59.378
```

- Specify source position manually (instead of using RA_OBJ and DEC_OBJ) for Timing mode:

```
emchain emframes:withsrccoords=Y  \
        emframes:srcra=204.6877 emframes:srcdec=-27.6984
```

- Run bad pixels finder on existing intermediate files:

```
rm badpix*.out.mos evmap.in.mos medmap.in.mos
emchain startfromodf=N clobber=N stopafterbadpixfind=Y
```

- Regenerate event files, using new bad pixels files and intermediate files (this will remove the intermediate files on successful output):

```
rm event*.out.mos
emchain startfromodf=N
```

- Relaunch **emchain** after an error which occurred on MOS 2, exposure 4, using already existing intermediate files:

```
rm PooooooooooM2S004*.FIT
emchain instruments=M2 exposures=4 clobber=N
```

3.7 How to deal with an error

If an error occurred, the very last output of emchain will be something like

```
emchain: BEWARE: One or more of the tasks ended in error !
```

You should then look into the log file for a specific message like

```
emchain: BEWARE: That task ended in error !
```

The associated messages may help you understand what happened. If the error occurred in the last exposure, the intermediate files will not be erased, you may also inspect them (if they were erased, rerun emchain for the particular instrument and exposure where the error occurred, setting `stoponerror`=Y).

If you think you have found a workaround (editing a file for example), you may relaunch the task which ended in error, duplicating the call written after 'CMD:' in the log file. You may also relaunch emchain from the intermediate files (see example above).

If the error is of general significance (not just a corrupted file at your site), feel free to send an Observation Report.

4 Parameters

This section documents the parameters recognized by this task (if any).

Parameter	Mand	Type	Default	Constraints

General parameters

ingtiset	no	string		none

user-supplied good time intervals

filteratt	no	boolean	no	yes/no

filter data on bad attitude

filterhk	no	boolean	no	yes/no

filter data on bad housekeeping

badpixfindalgo	no	string	EM	none

EP for **badpixfind**, EM for **embadpixfind**, NO for nothing

randomize	no	string	PET	

randomize multiswitch (P for position, E for energy, T for time). " for no randomisation at all

applyflaregti	no	boolean	no	yes/no

apply the proton flare GTI

Selection parameters

odf	no	string	SAS_ODF	none

input directory name (this is a standard SAS parameter)

exposures	no	list of strings	all	none

selected exposures (like S004, or simply 4 if non ambiguous)

instruments	no	list of strings	both	M1/M2

selected instruments

ccds	no	list of integers	all	1-7

selected CCDs

Parameters for running part of emchain

runccdloop	no	boolean	yes	yes/no

loop over CCDs

startfromodf	no	boolean	yes	yes/no

analyse raw ODF files

stopafterbadpixfind	no	boolean	no	yes/no

stop just after bad pixels detection to investigate

runevlistcomb	no	boolean	yes	yes/no

merge the CCD-specific events files

makeflaregti	no	boolean	yes	yes/no

build GTI for proton flare rejection

Parameters for debugging or calibration

rejectbadevents	no	boolean	yes	yes/no

reject events with any of the flags in `rejectionflag` set

rejectionflag	no	string	762aa000	none

hexadecimal representation of the flags triggering deletion

writeccdbackground	no	boolean	no	yes/no

save CCD background for offset calibration (one file per CCD)

processlowgain	no	boolean	no	yes/no

process data obtained in low gain read-out mode (as well as normal)

fulloutput	no	boolean	no	yes/no

keep all columns in event list (rather than only the ones in SSC products)

applyccdgti	no	boolean	yes	yes/no

apply the CCD-specific GTI

keepintermediate	no	boolean	no	yes/no

keep intermediate files (or remove them on output)

stoponerror	no	boolean	no	yes/no

stop at first error in task call

clobber	no	boolean	yes	yes/no

overwrite existing output files

Parameters for individual tasks

withatthkgen	no	boolean	no	yes/no

rerun **atthkgen**

atttol	no	real	0.05	> 0

tolerance for attitude filtering (degrees)

withbadpix	no	boolean	yes	yes/no

run **badpix**

withemevents	no	boolean	yes	yes/no

run **emevents**

withattcalc	no	boolean	yes	yes/no

run **attcalc**

withemenergy	no	boolean	yes	yes/no

run **emenergy**

lowenerbadpix	no	boolean	yes	yes/no

run **embadpixfind** a second time for energies < 500 eV

addvigweight	no	boolean	no	yes/no

run **evigweight**

globalflare	no	boolean	yes	yes/no

build single flare screening timeseries for the whole observation

flaretimebin	no	real	52.0	> 2.6

time bin for flare rejection (s)

flaremaxrate	no	real	2.0	> 0

threshold on rate of truncated singles for **tabgtigen** (cts/ks/arcmin^2)

The syntax taskname:parametername=soandso may be used to pass parameters to individual tasks called by **emchain**. See Sect. 3.5 for details.

Standard SAS parameters

Because **emchain** is a standalone Perl script, it does not deal with the standard SAS parameters (see **taskmain**) in exactly the same way as normal SAS tasks do:

- It does not support the '-d' (dialog) option. Actually **emchain** may not be called at all via the SAS GUI.
- It does not support the '-c' (noclobber) option because some constituent tasks do not work with that option set. That option is replaced by the `clobber` parameter specific to **emchain**.
- It emulates the '-h' (help), '-m' (manpage), '-p' (param), '-v' (version) parameters giving information on **emchain** itself.

- It passes all other standard SAS parameters to its constituent tasks. Specific OAL ('-o') and CAL ('-a','-f','-i') options are passed only to the tasks making use of the OAL or CAL, respectively. By default constituent tasks are called with '-w 10' (at most 10 warnings per task).
- All syntaxes (*e.g.* '-o your_odf_dir', 'odf=your_odf_dir', '--odf your_odf_dir') are supported.

Some SAS options are also interpreted at **emchain** level before being passed to its constituent tasks:

- '-V' (verbosity) is used in the same way as the SAS_VERBOSITY environment variable (see Sect. 3.1).
- '-o' (odf) is used to define the directory where the data resides in the same way as the SAS_ODF environment variable (see Sect. 3.1).
- '-i' (ccf) is used in the same way as the SAS_CCF environment variable to append a `CALINDEX` extension to the output file.

5 Errors

This section documents warnings and errors generated by this task (if any). Note that warnings and errors can also be generated in the SAS infrastructure libraries, in which case they would not be documented here. Refer to the index of all errors and warnings available in the HTML version of the SAS documentation.

odf *(error)*
No or non existing or empty input directory

odffiles *(error)*
No ODF-like file in input directory

lowGain *(warning)*
a CCD was operated in low gain mode. It is scientifically useless and **emchain** will normally not process it. If `processlowgain` is set, it will be processed but the flare screening light curve will be wrong unless this is the central CCD
corrective action: set `processlowgain`=Y if you want that CCD to be processed

badexposure *(warning)*
the event*.out files are incompatible with the requested exposure. The merged events list will not be created. This is possible only when `clobber`=N and event*.out files exist
corrective action: check the event*.out files and rerun

badinstrument *(warning)*
the event*.out files are not MOS files. The merged events list will not be created. This is possible only when `clobber`=N and event*.out files exist
corrective action: check the event*.out files and rerun

protonflare *(warning)*
The GTI fraction after flare screening is less than 10%. It is not applied
corrective action: reassess the flare GTI manually, or change `flaremaxrate` if you wish

noIN_FOV *(warning)*
the EXPOSUnn extensions of the merged events file do not contain the `IN_FOV` keyword. The flare detection cannot proceed
corrective action: rerun emchain on the ODF to regenerate the merged events file

badstring *(warning)*
a keyword does not have the expected length. It is truncated or padded with _. This should not happen on a regular ODF
corrective action: check whether a file is not corrupted in the ODF (you may know which file contains the offending keyword by looking at the messages), and that the substitution is all right

6 Input Files

1. event list files *_Mn*xxE.FIT (one per CCD/node and per exposure), straight from the ODF (n is 1 or 2, xx is IM or TI).
2. auxiliary files *_Mn*AUX.FIT (one per exposure), straight from the ODF.
3. offset/variance files *_Mn*OVE.FIT (one per CCD/node and per mode), straight from the ODF.
4. optionally, a user-supplied GTI file (entered through `ingtiset`) with `STDGTI` extension.
5. optionally, bad pixels files with `BADPIX` extension (one per CCD/node and per instrument). They are looked for in the current directory and their names must be P??????????DD????BADPIXij00.FIT, where i is the CCD number and j the node number. A file with matching observation and exposure fields is preferred, but any observation and exposure is accepted.

The structure of files in the ODF is described in [1]. The event and offset files in the ODF may be gzipped (end in .gz or .FTZ).

7 Output Files

1. event list files (one per instrument, exposure and per mode), as in the SSC Data Products ICD [2], except they are uncompressed. Their names are POOOOOOOOOODDUEEEMIEVLI0000.FIT (IMAGING mode) and POOOOOOOOOODDUEEETIEVLI0000.FIT (TIMING mode). If `addvigweight`=Y, a `WEIGHT` column is added.
2. timeseries files (one per instrument and per exposure) in RATE format of truncated single events per ks per arcmin^2, which are used to generate the good time intervals for flare screening. Their names are POOOOOOOOOODDUEEEFBKTSR0000.FIT. They contain the columns `RATE`, `ERROR`, `TIME` and `FRACEXP`. If `globalflare`=Y, a global timeseries is created in addition, named POOOOOOOOOOEMX000FBKTSR0000.FIT. In that one `FRACEXP` may be larger than 1.
3. good time interval files (one per instrument and per exposure) which may be used to select out proton flares in the data. Their names are POOOOOOOOOODDUEEEFBKGTI0000.FIT. If `globalflare`=Y, a single GTI file is created, named POOOOOOOOOOEMX000FBKGTI0000.FIT.
4. tracking history file POOOOOOOOOOOBX000ATTTSR0000.FIT generated by **atthkgen** (also useful for later tasks like **eexpmap**).

5. attitude GTI file POOOOOOOOOOOBX000ATTGTI0000.FIT generated by **tabgtigen** from the tracking history file.
6. HK GTI files (one per instrument) generated by **hkgtigen**. Their names are POOOOOOOOOODDX000HK_GTI0000.FIT.
7. Optionally (if `writeccdbackground`=Y), CCD background maps (one per CCD and per exposure) which may be used to check optical loading and the offsets map. Their names are POOOOOOOOOODDUEEECCDBKGij00.FIT, where i is the CCD number and j the node number.

8 Intermediate Files

1. `framenn.out.mos`: frame file for a given CCD/node in output of **emframes**, where nn is the same as the CCDNR column in the final output file (10*node + CCD).
2. `badpixnn.out.mos`: bad pixels file for a given CCD/node in output of **badpixfind** or **embadpixfind**.
3. `ccdgtinn.in.mos`: GTI file for a given CCD/node in output of **emframes**.
4. `extgti.in.mos`: external GTI file for a given instrument (merged from user, attitude and HK GTIs).
5. `hkgti.in.mos`: external GTI file aligned to frame readout boundaries for the current CCD/node in output of **gtialign**.
6. `gtinn.out.mos`: GTI file for a given CCD/node, intersection of the latter two.
7. `eventnn.in.mos`: input event list file for a given CCD/node, with `BADPIX` extension appended by **badpix**.
8. `eventnn.out.mos`: event list file for a given CCD/node in output of **emevents**, with `BADPIX`, `OFFSETS0`, `OFFSETS` and `EXPOSURE` extensions, propagated through **attcalc** and **emenergy**.
9. `merged.img.mos`: merged event list in IMAGING mode in output of **evlistcomb**.
10. `merged.tim.mos`: merged event list in TIMING mode in output of **evlistcomb**.
11. `merged.truncated.mos`: selection of events with truncated energy (for flare screening).

9 Algorithm

```
subroutine emchain

   Read parameters.

   if startfromodf then call processOdf
   else if runevlistcomb then call mergeEvents

   if makeflaregti then
      Loop over the *MIEVLI0000.FIT files
         call makeFlareTS
         if globalflare  call mergeTS  else  call tsToGTI
      end loop
      if globalflare  call tsToGTI
```

```
   endif

   Loop over the *MIEVLI0000.FIT files
      if withflaregti  call applyFlareGTI
      if addvigweight  call evigweight ineventset=eventfile
   end loop

   if not keepintermediate then  rm -f *.in.mos *.out.mos merged.*.mos

end subroutine emchain

subroutine processOdf

   Set SAS_ODF to odf (for OAL).

   Call atthkgen if no tracking history file or withatthkgen is true.
   Call tabgtigen with expression='!isNull(DAHFPNT) && DAHFPNT<atttol'
   Call hkgtigen if no HK GTI is present.
   gtiin = 'extgti.in.mos'
   Call gtimerge gtitable=$gtiin on attitude GTI (if filteratt),
                HK GTI (if filterhk) and user GTI (if present)

   Look in odf for ODF files pertaining to selected instrument,
   ending in IME, TIE, RIE or CTE.FIT (possibly .gz or .FTZ).
   Deduce all exposures present.

   Loop over selected exposures
      if badpixfindalgo ne 'NO' then
         call ccdLoop (forbadpixfind=Y)
         call badpixLoop (no flare screening)
         call mergeEvents
         call ebadpixpixupdate
         call buildFlareGTI
         call badpixLoop (with flare screening and low energy run)
      endif

      call ccdLoop (forbadpixfind=N)
      if runevlistcomb then call mergeEvents
   end loop over exposures

end subroutine processOdf

subroutine ccdLoop

   Loop over selected CCDs and nodes

      Identify event file name event0 ending in IME, TIE, RIE or CTE.FIT.
      Deduce ccd and node.

      aux = substring($event0 - last 9 characters) // '00AUX.FIT'
      gtiout = 'gti' // node // ccd // '.out.mos'
      frameout = 'frame' // node // ccd // '.out.mos'
      emframes auxiliaryset=frame.in odfeventset=$event0 \
               frameset=$frameout writegtiset=Y outgtiset=$gtiout
      if ($gtiin exists) then
```

```
         flagbadtimes=Y ingtiset=$gtiin

      if (not processlowgain and GAIN_CCD='LOW') next

      bad = 'badpix' // node // ccd // '.out.mos'
      if $bad does not exist then
         bad = 'P*' // instrument // '*BADPIX' // ccd // node // '00.FIT'

      if withbadpix and not forbadpixfind then
         badpix eventset=$event0 withoutset=Y outset=event.in windowfilter=Y
         if ($bad exists) then
            getuplnkbadpix=Y getotherbadpix=N getnewbadpix=Y badpixset=$bad
      else
         cp $event0 event.in.mos
      endif

      eventout = 'event' // node // ccd // '.out.mos'
      off = odf // * // instrument // * // ccd // node // 'OVE.FIT'
      emevents withframeset=Y frameset=$frameout odfeventset=event.in \
            eventset=$eventout
      if ($off exists) then  withoffvarsets=Y offvarsets=$off
      if not randomizeP then  randomizeposition=N
      if     randomizeT then  randomizetime=Y
      if forbadpixfind then  analysepatterns=N flagbadpixels=N \
                             splitdiagonals=N randomizeposition=N

      if ($gtiin exists) then
         gtialign gtitable=$gtiin:STDGTI eventset=$eventout \
                  outset=hkgti.in.mos
         extname = 'STDGTI' // node // ccd
         gtimerge tables="hkgti.in.mos ccdgti.in.mos" mergemode=and \
                  gtitable=$gtiout:$extname

      if withattcalc and not forbadpixfind then
         attcalc eventset=$eventout

      emenergy ineventset=$eventout
      if not imaging mode then  getccdbkg=N
      if not randomizeE then    randomizeenergy=N
      if forbadpixfind then     correctcti=N correctgain=N randomizeenergy=N
      bkg = 'P'//obsid//instrument//expid//'CCDBKG'//ccd//node//'00.FIT'
      if writeccdbackground then backgroundset=$bkg

      if rejectbadevents or forbadpixfind then
         evselect table=$eventout destruct=Y keepfilteroutput=Y \
                  expression="(FLAG & 0x$rejectionflag) == 0"

   end loop over CCDs and nodes

end subroutine ccdLoop

subroutine badpixLoop

   Loop over selected CCDs and nodes
```

```
      eventout = 'event' // node // ccd // '.out.mos'
      bad = 'badpix' // node // ccd // '.out.mos'

      evselect table=$eventout withfilteredset=Y filteredset=event.in.mos  \
               keepfilteroutput=Y  destruct=Y writedss=Y updateexposure=Y \
               expression="TIME in GTI($gtiflare)"

      if badpixfindalgo == 'EM' then
         emeventsproj eventset=event.in.mos rejectbadevents=Y  \
                      evimageset=evmap.in.mos
         embadpixfind evimageset=evmap.in.mos badpixset=$bad

         if (lowenerbadpix) select PHA < 150 and run again incrementally
      else
         badpixfind eventset=event.in.mos thresholdlabel=rate \
                    badpixset=$bad  \
                    hithresh=0.005 narrowerthanpsf=3.0 backgroundrate=1.E-5
      endif

   end loop over CCDs and nodes

   if stopafterbadpixfind  stop

end subroutine badpixLoop

subroutine mergeEvents

   evlistcomb eventsets='event*.out.mos' imagingset=merged.img.mos \
              timingset=merged.tim.mos maintable='EVENTS OFFSETS'
   if fulloutput then
      emosimgcolnames="TIME RAWX RAWY DETX DETY X Y PHA PI FLAG PATTERN
          FRAME ENERGYE1 ENERGYE2 ENERGYE3 ENERGYE4 PERIPIX OFFSETX OFFSETY"
      emosimgcoltypes="double int16 int16 int16 int16 int32 int32 int16
          int16 int32 int8 int32 int16 int16 int16 int16 int8 int16 int16"
   endif

   if applyccdgti then
      Loop over gti*.out.mos files
      expr = expr // '|| (CCDNR==' //node//ccd // ' && GTI($gtiout,TIME))'
   endif

   eventim = 'P' // obsid // instrument // expid // 'MIEVLI0000.FIT'
   evselect table=merged.img.mos withfilteredset=Y filteredset=$eventim \
            expression=$expr destruct=Y keepfilteroutput=Y

   fparkey "EPIC MOS IMAGING MODE EVENT LIST" $eventim[0] CONTENT \
      add=Y insert=DATE"
   fappend $sasccf[CALINDEX] $eventim

   if (merged.tim exists) then
      eventti = 'P' // obsid // instrument // expid // 'TIEVLI00001.FIT'
      evselect table=merged.tim.mos withfilteredset=Y filteredset=$eventti \
               expression=$expr destruct=Y keepfilteroutput=Y
      fparkey "EPIC TIMING MODE EVENT LIST" $eventti[0] CONTENT \
         add=Y insert=DATE"
```

```
      fappend $sasccf[CALINDEX] $eventti
   endif

end subroutine mergeEvents

subroutine makeFlareTs
   expr = "(PATTERN==0) && ((FLAG & 0x762b8000) == 0) && #XMMEA_22"
   flarets = 'P' // obsid // instrument // expid // 'FBKTSR0000.FIT'
   check that expected counts per bin in quiet conditions is > 10,
      otherwise increase $flaretimebin
   evselect table=$eventim expression=$expr updateexposure=N \
            withrateset=Y rateset=$flarets timebinsize=$flaretimebin \
            timecolumn=TIME maketimecolumn=Y makeratecolumn=Y
   add FRACEXP column (currently done by looking at full timeseries)
   divide by CCD area (IN_FOV keyword) and FRACEXP
end subroutine makeFlareTs

subroutine mergeTs
   globts = 'P' // obsid // 'EMX000FBKTSR0000.FIT'
   if $globts exists then
      multiply by CCD area and FRACEXP
      merge columns of $flarets and $globts:
      RATE    = RATE1 + RATE2
      ERROR   = SQRT(ERROR1**2 + ERROR2**2)
      FRACEXP = FRACEXP1 + FRACEXP2
      divide by CCD area and FRACEXP
   endif else then
      cp $flarets $globts
   endelse
end subroutine mergeTs

subroutine tsToGTI
   gtiflare = timeseries prefix //  'FBKGTI0000.FIT'
   tabgtigen table=$flarets expression="RATE<$flaremaxrate" \
             gtiset=$gtiflare
   evselect table=$gtiflare writedss=N updateexposure=N keepfilteroutput=Y \
            expression="(STOP - START) > 1.5*$flaretimebin"
end subroutine tsToGTI

subroutine applyFlareGTI
   expr = "GTI($gtiflare,TIME)"
   evselect table=$eventim destruct=Y keepfilteroutput=Y expression=$expr
   evselect table=$eventti destruct=Y keepfilteroutput=Y expression=$expr
end subroutine applyFlareGTI
```

10 Comments

The script does not use the SAS' parameter and error interfaces, to allow easy modifications by users.

11 Future developments

The chain will adapt to the evolution of its constituents and to the organisation of the pipeline.

References

[1] ESA. XMM Interface Control Document: Observation and Slew Data Files (XSCS to SSC) (SciSIM to SOCSIM). Technical Report XMM-SOC-ICD-0004-SSD Issue 2.5, ESA/SSD, June 2000. Found at the URL: `ftp://astro.estec.esa.nl/pub/XMM/documents/odf_icd.ps.gz`.

[2] SSC. XMM Survey Science Centre to Science Operations ICD for SSC Products. Technical Report XMM-SOC-ICD-0006-SSC Issue 2.1, SSC, Mar 2000.