# badpix

February 1, 2016

**Abstract**

The **badpix** task collects together pixels identified as 'bad' (whether 'hot', 'dead', 'flickering' etc.) and therefore as pixels containing no useful data. A badpix table, containing information on the position and the type of defect of the bad pixels, is produced as an extension to either the input raw events file or a copy of this file.

## 1 Instruments/Modes

| Instrument | Mode |
|---|---|
| EPIC MOS | ALL |
| EPIC PN | ALL |
| (RGS) | (SPECTROSCOPY) (see comments) |

## 2 Use

| | |
|---|---|
| pipeline processing | yes |
| interactive analysis | yes |

## 3 Description

**badpix** collects together pixels identified as containing no useful data, and uses this information to produce a badpix extension table to either the input raw events file or a copy thereof. Note that no alteration of the actual events extension is performed.

A very important point regarding the bad pixels is that there are essentially *three separate* (though not exclusive) sets of bad pixels that must be dealt with. These are (1) the bad pixels uplinked to the satellite and eliminated on-board, (2) the bad pixels identified in the CCF but not uplinked, and (3) the additional bad pixels associated with the particular observation in question. Sets (1) and (2) are contained within the CCF (and in the case of MOS, within the ODF iteself - see below), and **badpix** deals with these first two sets of bad pixels, though, if a file exists containing any badpix set (3) information (created using the task **badpixfind**), then this information is accessed in addition (if requested).

What **badpix** does is to create the badpix extension to the input raw events file (or to a copy thereof), and fill this with the information regarding any or all of the badpix sets (1), (2) and (3) (from the **badpixfind** output file).

The user will be able to specify which set(s) of bad pixels should be used to construct the badpix extension (whether just (1), just (2), just (3), or any combination of these). Once this has been decided upon, the badpix extension is constructed and appended to the input raw events file (or a copy), and filled with the relevant information.

In cases where a **badpixfind** file is accessed (badpix set (3)), and either or both of the CCF badpix sets (1 and/or 2), duplicity checking and purging of the **badpixfind** pixels is performed (see comments).

Command switches also exist to append a purely empty `BADPIX` table, or to append a `BADPIX` table containing only those pixels within the input file X/Y window (see parameters and comments).

In the case of MOS, the task accesses the uplinked (set 1) bad pixels from the actual ODF data files of the observation in question. These are then used in preference to any uplinked (set 1) bad pixels found within the CCF. Non-uplinked (set 2) MOS bad pixels are taken from the CCF.

# 4 Parameters

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

| **Parameter** | Mand | Type | Default | Constraints |
|---|---|---|---|---|
| **eventset** | yes | string | | |
| input raw events set | | | | |
| **getuplnkbadpix** | yes | logical | Y | Y/N |
| get uplinked bad pixels (from CCF)? | | | | |
| **getotherbadpix** | yes | logical | Y | Y/N |
| get non-uplinked bad pixels (from CCF)? | | | | |
| **getnewbadpix** | yes | logical | N | Y/N |
| get new pixels from task **badpixfind**? | | | | |
| **badpixset** | no | string | badpixfind.fits | |
| output from **badpixfind** (for getnewbadpix = Y) | | | | |
| **emptyextension** | no | logical | N | Y/N |
| Create an empty `BADPIX` extension? | | | | |
| **windowfilter** | no | logical | N | Y/N |
| Just get pixels within input file X/Y window? | | | | |
| **withoutset** | no | logical | N | Y/N |
| Create new output file (`BADPIX` extension is appended to input if false) | | | | |
| **outset** | no | string | out.fits | |
| Name of optional output file (if `withoutset`=Y) | | | | |

# 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.

**badExtension** *(error)*
No, invalid or incorrectly named extension in input events file

**badInstrument** *(error)*
Instrument not supported

**noBadpixModes** *(error)*
No **badpix** modes chosen

**BPTcode** *(warning)*
BadPixelTable code in PAH.FITS and BADPIX.CCF differ
*corrective action:* continue, probably the BADPIX.CCF is incorrect (too old), bad pixel positions may be wrong

**badBadpixfindFile** *(warning)*
**badpixfind** file is incompatible with input file
*corrective action:* Continue, with `getnewbadpix`=N

**overwriteBadpix** *(warning)*
`BADPIX` extension already exists - overwriting
*corrective action:* Continue, with warning message

# 6 Input Files

1. Raw event list from **epframes** or **emevents**. Attributes read: `INSTRUME` `QUADRANT` `CCDID` `CCDNODE` `WINDOWX0` `WINDOWY0` `WINDOWDX` `WINDOWDY`.
2. Output file from **badpixfind** (optional). Attributes read: `TELESCOP` `INSTRUME` `CCDID` `EXP_ID` `OBS_ID` `QUADRANT` `CCDNODE`

# 7 Output Files

1. File as input (either the actual input file or a copy - see parameters) with a bad pixel extension (named `BADPIX`) giving the RAWX/RAWY and extent in the Y direction of each bad pixel (or strip thereof), the type of fault present, and the source of the pixel (i.e. whether from badpix sets 1, 2 or 3) (column names are `RAWX`, `RAWY`, `YEXTENT`, `TYPE` and `BADFLAG`). This file is next accessed by **epevents** or **emevents**.

# 8 Algorithm

```
subroutine badpix
```

```
  * get input events, if required, make a copy of this file
  * get instrument, datamode, ccdid and node from events file
  * get badpix modes (1,2,3)?
  * enter CCF, and get bad pixel list (from CCF)
  * if mode3 (getnewbadpix), find size of mode3 file (output from
    badpixfind), and check if compatible with events file
  * check and purge mode3 file for double entries
  * get total size of badpix table, combining modes 1,2 and 3
  * check if badpix extension already exists, and if not, allocate columns
    for badpix information
  * fill columns with relevant mode1/mode2 (i.e.\ getuplnkbadpix/
                                                 getotherbadpix) information
  * as above, for mode 3 pixels (if required)
  * check mode 3 pixels against information in CCF
  * purge mode 3 entries for doubles in CCF
  * check each pixel for common (though not identical) entries
  * if filterwindow mode, filter bad pixels for entries within X/Y window
  * add extension to input events file, fill in badpix information
  * add attributes (instrument, ccdid etc) and history information

end subroutine badpix
```

# 9 Comments

- The task was originally designed to handle RGS data also. The `BADPIX` extension here contains the CHIPX/CHIPY chip-orientated co-ordinates of the bad pixels. Though this functionality still exists, it is recommended to use the task **rgsbadpix**.
- The task checks for duplicity between any **badpixfind** input and bad pixels contained within the CAL. Irrespective of whether CCF bad pixels are required to be input into the `BADPIX` extension or not, the task gives general information as to CCF/**badpixfind** double entries. If entries from the CCF and from **badpixfind** covering the same coordinates are required to be input into the `BADPIX` extension, then the **badpixfind** entries are split and purged such that no RAWX/RAWY coordinate has more than one entry (i.e. the CCF entries take precedent). Also the task checks whether the **badpixfind** entries themselves have any overlapping pixels. If so, the **badpixfind** data are purged, precedent being given to hot pixels over dead pixels (over flickering pixels).
- The task, via the parameter `emptyextension`=Y, produces an empty `BADPIX` extension. Also, via the parameter `windowfilter`=Y, the task filters the bad pixels for those within the input file WINDOW range (entries are adjusted if they lie across the WINDOW border). The default values for both parameters are 'N'.
- Information from the hardware groups is required regarding how the bad pixels will be handled on board in timing and burst mode. The **badpix** task at the ground could, for example, use the same badpix sets (1 and 2), as for the imaging modes, and add them as an extension to the raw event file. Event flagging then has to be dealt with in the events tasks.

# 10 Future developments

Future calibration developments may need to be incorporated into the task.

# 11 Examples

- *badpix eventset=events.dat getnewbadpix=Y badpixset=badpixfind.fits* (badpix extension, with uplinked and non-uplinked CCF bad pixels [both by default] and within the file badpixfind.fits, appended to input events file).
- *badpix eventset=events.dat getuplnkbadpix=N getotherbadpix=N getnewbadpix=Y badpixset=badpixfind.fits withoutset=Y outset=newevents.dat* (badpix extension, with just the bad pixels in badpixfind.fits appended to a copy of the input events file [named newevents.dat]).

# References