r e p o r t

Kavli Institute, MIT NE83-545
Building 300, One Kendall Square
Cambridge MA, 02139–4307
Tel: +1-617-253-7277
Fax: +1-617-253-8084

To: ACIS Science Operations Team

From: Peter Ford, NE83-545 <pgf@space.mit.edu>

Date: April 22nd 2019

Subject: ACIS Event Filtering (v 1.0)

1. Introduction

The ACIS Back-End Processor (BEP) receives candidate events from the front-end processors (FEPs) and performs a series of tests to reject candidates based on fields in the parameter block ("pBlock") that controls the observation, and in an optional window block ("wBlock") referenced by the pBlock. This report details the filtering algorithms and the order in which they are applied.

There are two types of pBlock: `loadTeBlock` for timed-exposure observations and `loadCcBlock` for continuous-clocking observations, and two corresponding wBlocks: `load2dBlock` and `load1dBlock`. Their sub-fields are defined in section 4.1.4 of the ACIS Software User's Guide (see §6, below). In practice, the binary blocks are created by the *bcmd* command whose on-line manual ("*man bcmd*") defines the structure and nomenclature of its ASCII input. The examples in this report will use this format.

2. Event Filtering

Each event candidate is subject to three types of filtering, in the following order. As soon as an event is ignored, any remaining filter operations will be skipped.

a. Event Thresholds

Each CCD is divided into 4 quadrants of 256 pixel columns each and each group is amplified and digitized independently. FEPs search for event candidates on the basis of the values of `fep`*n*`EventThreshold[`*i*`]` in the `loadTeBlock` and `loadCcBlock` commands, where *n*=0...5 denotes the FEP and *i*=0...3 denotes the quadrant. The FEP examines each pixel, subtracts its corresponding bias map value and uses frame-averaged overclock values to correct for instrument drift. If the resulting "reduced pixel" value is not less than the quadrant's `fep`*n*`EventThreshold[`*i*`]` value and is a local maximum relative to the 8 surrounding reduced pixels, the event is passed to the BEP for further examination. If more than one of the 9 pixels shares the same maximum value, only the first to be in the center will be passed to the BEP.

The FEP thresholds filter out many charge patterns that are most unlikely to result from x-ray events, but must be set low enough so as not to exclude soft x-rays. In practice, the chosen values, 38 ADU for front-illuminated CCDs and 20 ADU for back-illuminated ones, are ideal for most observations and are expected by the CXO processing pipelines. The only exception is when observing extended targets that are bright in the near infrared, *e.g.*, Venus, Jupiter and Saturn, where sufficient optical light is transmitted through the ACIS blocking filters to add 10-30 ADU to each pixel within the planetary disk for 3 second exposures. It has become customary to increase the thresholds by a similar amount when making these observations, but the resulting fluxes must be corrected for events lost near the limbs of the planets.

b. Amplitude

The pulse-height amplitude (*pha*) of the event candidate is computed by taking the 3x3 reduced pixels at the center of the event, and summing those that contain significant charge. If the resulting *pha* is less than the `lowerEventAmplitude` field in the pBlock, or equal to or greater than its `lowerEventAmplitude+ eventAmplitudeRange`, the event will be ignored and the `discardEventAmplitude` in the current *exposure* packet will be incremented.

c. Grade Code

In a timed exposure observation, 3x3 pixels of each event candidate are converted into an 8-bit integer Grade Code. Consider the table on the left representing the 3x3 reduced pixels with row numbers increasing from

32	64	128
8	0	16
1	2	4

bottom to top and column numbers increasing from left to right. The grade code is the sum of the values in each square of the table when value of the corresponding reduced pixel is not less than `fep`*n*`splitThreshold[`*i*`]`, where *n*=0...5 denotes the FEP and *i*=0...3 the quadrant (see §2a above).

Within the BEP, the grade code defines an offset into the 256-bit `gradeSelections` field in the `loadTeBlock`. If the bit at that location is zero, the event is ignored and the `discardGrade` field in the current exposure packet will be incremented. The `gradeSelections` item in *bcmd* format breaks the array into eight 32-bit words, with the least significant word first and, within each word, least significant bit first. Hence the default definition in current pBlocks,

```
gradeSelections =     0xfeffffff 0xffffffff 0xfffffffb 0xfffff7ff
                      0xffffffff 0xffffffff 0xffbfffff 0x7fffffff
```

instructs the BEP to accept all grades except 24, 66, 107, 214 and 255. For a fully discussion of grade codes and their uses, consult section 6.15 of the Chandra Proposers' Observatory Guide (see §6, below).

In 1x3 continuous clocking mode, there are only 2 pixels surrounding the center one, so the grade consists of a 2-bit integer and the `gradeSelections` field in the `loadCcBlock` is therefore 4 bits in length. In 3x3 continuous clocking mode, the grade code is an 8-bit integer formed in the same manner as in timed exposure mode, but the 4-bit `discardGrade` field is interpreted as an index into an array of 16 elements, each representing a 256-bit `gradeSelections` array, initialized by the "cc3x3" patch that implements this mode. The assignments in Revision A of the patch are as follows:

`discardGrade`	Description
0	Reject all grades
1	Reject ASCI grades 1, 2, 3, 4, 5, 6, 7
2	Reject ASCA grades 1, 5, 6, 7
3	Reject ASCA grades 1, 5, 7
4	Undefined
5	Undefined
6	Undefined
7	Reject ACIS flight grades 24, 66, 107, 127, 214, 223, 248, 251, 254, 255
8	Reject ACIS flight grades 24, 107, 127, 214, 223, 248, 251, 254, 255
9	Reject ACIS flight grades 24, 66, 107, 214, 248, 255
10	Reject ACIS flight grades 24, 66, 107, 214, 255
11	Reject ACIS flight grades 24, 107, 214, 248, 255
12	Reject ACIS flight grades 24, 107, 214, 255
13	Reject ASCA grade 7
14	Reject ACIS flight grade 255
15	Accept all grades

d. Window Filters

If the `windowSlotIndex` field of the pBlock contains a number between 0 and 4, the BEP will use the wBlock in that particular memory slot (1- or 2-dimensional according to whether the pBlock reflects timed-exposure or continuous clocking mode) to submit each event candidate to window filtering, as described in the following section. If this causes the event to be ignored, the `discardWindow` field in the current exposure packet will be incremented.

3. Window Filtering

A wBlock contains one or more window filters, to be applied to each event candidate in the order in which they appear in the wBlock definition. Once a candidate matches the `ccdId` and pixel location criteria, it is accepted or rejected on the basis of the `lowerEventAmplitude`, `eventAmplitudeRange` and `sampleCycle` values of that window only, ignoring the remaining windows. The location of an event candidate is determined by the `ccdRow` and `ccdColumn` values of the center of the 3x3 pixel array, each running from 0 to 1023 except in continuous-clocking mode where `ccdRow` is replaced by `transferRow` and is ignored by the 1-D window filters.

Each window is assigned a `sampleCycle` value from 0 through 255. Zero means that if this is the first window to apply to a particular event candidate, the event must be discarded, whatever its `pha` value. Otherwise, a value of `n` means that, if the candidate falls within the `pha` range specified by the values of `lowerEventAmplitude` and `eventAmplitudeRange`, it will only be accepted if it is the `n`'th candidate to fall within this window that passes the `pha` test. The first candidate to survive the `pha` test is always accepted. The function of a `sampleCycle` greater than 1 is to restrict the number of events coming from a given area that might otherwise saturate the ACIS output telemetry allocation.

The following example shows the *bcmd* definition of a 2-dimensional wBlock used to filter events from a timed exposure observation. The `windowBlockId` is a unique 32-bit integer, "`0xnnnnnvvv`", where "`vvv`" is a revision number. In the Chandra Offline System, the command packet that defines and loads the block will be named "`W2nnnnnvvv`", so in this example it would be `W200133014`, its `commandIdentifier` will be 15579, and it will be loaded into 2D-window slot number 4.

```
load 15579 window2d 4 {
      windowBlockId               = 0x00133014
      window = {
             ccdId                      = 2
             ccdRow                     = 21
             ccdColumn                  = 0
             width                      = 1023
             height                     = 199
             sampleCycle                = 1
             lowerEventAmplitude        = 0
             eventAmplitudeRange        = 65535
      }
      window = {
             ccdId                      = 2
             ccdRow                     = 0
             ccdColumn                  = 0
             width                      = 1023
             height                     = 1023
             sampleCycle                = 0
             lowerEventAmplitude        = 0
             eventAmplitudeRange        = 65535
      }
}
```

This wBlock consists of two window filters that operate on events from `ccdId` 2, *i.e.*, the I2 chip. The first window applies to pixels in rows 21 through 220 (21+199) and all columns (0 through 1023). Within this region, all `pha` energies will be accepted (0 through 65535) and all surviving pixels will be accepted. The second window applies to all rows (0 through 1023), all columns (0 through 1023) and all energies, but a zero value of `sampleCycle` causes all events to be rejected. So the net effect of the two windows in this wBlock

will be to restrict events from CCD_I2 to rows 21 through 220 and all columns. Note that `width` and `height` define one less than the number of columns and rows in the window. Since an observation can use only one window block at a time, window filtering will not be applied to any other CCDs used in this particular run.

Continuous clocking observations do not distinguish the row of an event. They use one-dimensional window filters defined by wBlocks that are similar to the two-dimensional ones but omit any mention of rows or heights. Here is the *bcmd* definition of a one-dimensional block named `W10011C014`:

```
load 13396 window1d 4 {
      windowBlockId              = 0x0011c014
      window = {
             ccdId                    = 7
             ccdColumn                = 154
             width                    = 99
             sampleCycle              = 10
             lowerEventAmplitude      = 20
             eventAmplitudeRange      = 3250
      }
}
```

4. Filtering Logic

The following algorithm demonstrates how an event candidate identified in a FEP is filtered by the BEP, first by the values of fields in the `pBlock` and then by windows in the `wBlock`, if any.

```
  REJECT if event's pha is outside pBlock's energy range
  REJECT if event's grade isn't marked in pBlock's gradeSelections array

  for each window (if any) in the wBlock
      if the event comes from this CCD and lies within this window's bounds
          REJECT if this window's sampleCycle is zero
          REJECT if the event's pha is outside this window's energy range
          increment this window's event counter
          ACCEPT if (event counter - 1) modulo this window's sampleCycle is zero
          REJECT
      end if
  end for

 ACCEPT
```

Note that `sampleCycle=0` forces an event to be rejected before testing its `pha` value while `sampleCycle>0` only affects events that have passed the `pha` test.

In a more formal way, the code below shows how `pBlock` and `wBlock`, represented by C++ structures of unsigned integer variables and arrays, control the filtering. The `exposure` structure represents the ACIS packet that reports the current exposure, and `win` is one of the windows in `wBlock`. Each window's internal `eventCnt` variable is initialized to zero at the start of each run and is incremented if and only if the event candidate passes all other tests for that particular window.

```
int filterEvent(int ccd, int row, int col, int pha, int grade) {
    if (pha < pBlock.lowerEventAmplitude) {
        exposure.discardEventAmplitude++;
        return REJECT;
    } else if (pha >= pBlock.lowerEventAmplitude+pBlock.eventAmplitudeRange) {
       exposure.discardEventAmplitude++;
        return REJECT;
    } else if (pBlock.gradeSelections[grade].gradeSelectValue != 1) {
       exposure.discardGrade++;
        return REJECT;
    }
```

```
    for (int ii = 0; ii < nwindows; ii++) {
         Window win = wBlock.windows[ii];
         if (ccd == win.ccdId &&
             row >= win.ccdRow && row <= win.ccdRow + win.height &&
             col >= win.ccdColumn && col <= win.ccdColumn + win.width) {
             if (win.sampleCycle == 0) {
                 exposure.discardWindow++;
                 return REJECT;
             } else if (pha < win.lowerEventAmplitude) {
                 exposure.discardWindow++;
                 return REJECT;
             } else if (pha >= win.lowerEventAmplitude+win.eventAmplitudeRange) {
                 exposure.discardWindow++;
                 return REJECT;
             } else if ((win.eventCnt++ % win.sampleCycle) == 0) {
                 exposure.eventSent++;
                 return ACCEPT;
             } else {
                 exposure.discardWindow++;
                 return REJECT;
             }
         }
     }
     exposure.eventSent++;
     return ACCEPT;
}
```

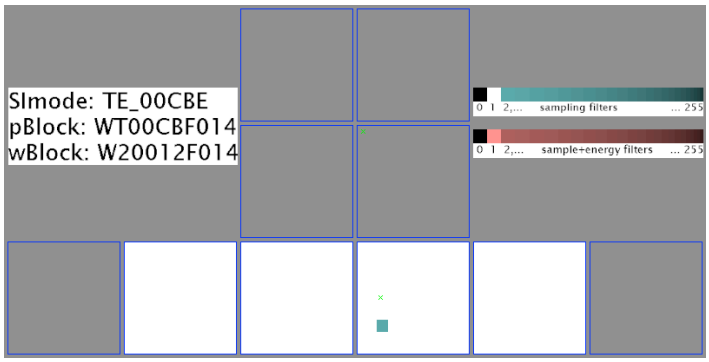
Note that if an event is rejected because its `pha` falls outside the energy limits set in the `pBlock`, it is the `discardEventAmplitude` that is incremented, but if an event passes that `pha` test but is rejected by the `pha` limits of the first `wBlock` that includes that event's location, it is `discardWindow` that is incremented.

5. Visualization

To assist in preparing window filters, the ACIS EGSE libraries include the *lwin* script that creates a graphical representation from one or more entries in the ACIS offline tables. It can display the result of applying individual window blocks or, more usefully, SIMODEs, since the latter include the parameter block that defines the set of CCDs to be used, along with any possible window block and the range of `pha` values accepted. In the following example, *lwin* writes a 1/8 scale PNG-format graphic of the window coverage of SIMODE "`TE_00CBE`" into the file "*/data/wins/TE_00CBE.png*".

```
lwin -o /data/wins -f png -s3 TE_00CBE
```

lwin possesses many command-line options, including the ability to create graphics in numerous formats and scales for all SIMODEs in an ACIS table. The fully-decorated version of *TE_00CBE.png* is shown on the right. The small green crosses mark the nominal ACIS-I and ACIS-S aimpoints. This observation used 4 CCDs, S1 through S4, with a small window filter centered on the aimpoint with `sampleCycle=10` to restrict zeroth order events from an isolated bright target.



The window was defined by the *bcmd* instructions below. Since the `pha` limits in `lowerEventAmplitude` and `eventAmplitudeRange` were no more restrictive than those in the accompanying `pBlock`, the window area in the graphic is colored blue-green, where its density relates to its `sampleCycle` value according to the scale on the right. Had the window's `pha` limits been more restrictive than those in `WT00CBF014`, that SIMODE's `pBlock`, the window would have been displayed in shades of pink. Note that a `sampleCycle=1` window is displayed in white unless it restricts `pha`, in which case it will be bright pink.

```
window = {
    ccdId                   = 7
    ccdColumn               = 154
    width                   = 99
    sampleCycle             = 10
    lowerEventAmplitude     = 20
    eventAmplitudeRange     = 3250
}
```

Window graphics for all SIMODEs in the ACIS archive can be viewed at the ACIS web site "*http://acis.mit.edu/asc/*". Typing the name in the "*PacketId/siMode*" window in the "*ACIS Object Catalog Search*" section will display the contents in ASCII. Then clicking the "*ACIS Command Tag*" title will display the graphic. Also, typing an OBSID in the "*Obsid in OCAT*" will display details of that observation and clicking on the thumbnail graphic of the CCD arrangement will display the detailed graphic, as shown above.

6. References

1. ACIS EGSE User Commands, MIT, revised January 31, 2019.
2. ACIS Science Instrument Software User's Guide, MIT 36–54003, NAS8–37716, Revision A (1999).
3. ACIS IP&CL Structures, MIT, revised July 28 2014.
4. ACIS IP&CL Structure Definition Notes, MIT 36–53204.0204, Revision N (2003).
5. The *Chandra* Proposers' Observatory Guide, Section 6.22, Revision 21.0, December 2018.

7. Glossary

ADU	Analog to Digital Unit: the approximate energy of an ACIS pixel in units of ~3.5 eV
bcmd	The ASCII representation of a command to BEP flight software
BEP	ACIS back-end processor controlling ACIS hardware and conducting science observations
Chip	One of the 10 x-ray sensitive CCDs in the ACIS focal plane, numbered $0 \leq$ `ccdId` ≤ 9
EGSE	Electrical ground support equipment; especially, in this document, ground support software
FEP	One of 6 ACIS front-end processors, each passing event candidates from a CCD to the BEP
Grade	The pattern of charge distribution in the pixels neighboring the center of an event candidate
`pBlock`	A science run parameter block controlling an observation
`pha`	Pulse-height amplitude: a crude measure of the energy of an x-ray candidate in ADUs
SIMODE	The name assigned to the set of BEP commands that configure and start an observation
`wBlock`	A science run window-filter block (optionally) defining spatial filters