RXTE GOF: DDL, SEFILTER, and Event Mode Data

Recipes from the RXTE Cook Book

This is a general recipe discussing PCA event mode data, the data descriptor language (DDL), and the tool **sefilter**. All of this information can be found in the technical appendix (Appendix F), the XFF Documents, the fhelp, other recipes, or the *ABC Guide* and is summarized here in a more compact and perhaps clearer form.

- **Event words and the Data Descriptor Language**
- **Determining what you can filter**
- **Using sefitler**
    - **GoodXenon example**
    - **Generic event example**
- **A note on strategy**

In science event data, all of the information is encoded in the individual bits of the event word. Different event modes encode the information in different ways expressed using the Data Descriptor Language (DDL; see the *ABC Guide*). This recipe summarizes common examples and how to use them.

The first thing that is important to realize is that, since the data are stored as a series of bits, a "bit-mask" must be developed which is appropriate for the file that is being processed. This bit-mask is of the form "Column name, relational operators, Bit-mask" (e.g. Event == b1xxxxxxxxxxxxxxx, with the "bit-mask" being denoted from other types of values by the presence of a "b" in front of the bit-mask.) You do not need to make this expression yourself; the script **sefilter** will make it for you if you know what values you want in different parts of the event word. It will give you a token expressed in the DDL and ask you for a value. It will then construct an expression in the DDL which is passed to **sebitmask** which actually constructs the bitmask. Sefilter then gives you options for using the bitmask depending on its form. Neither the extractor nor fselect is capable of determining if a bitmask is sensible, however, so you need to understand the DDL in order to be sure you get what you intend.

Event Words and The Data Descriptor Language:

There are two keywords in the header of your FITS event list which tell you what type of events are in the list and how they are formatted. Use **fdump** or **fkeyprint** to see the TDDES2 and TEVTB2 keywords in the bintable extension. Both will be expressed in the DDL, which is quickly summarized here.

In brief, an event is represented by a string of ones and zeros, or an event word, which can be split into sections called tokens. For instance, one part of the data descriptor is often the D[0:4]{3} describing detector identification. This notation tells you that the token uses three bits to represent a denary value between 0 and 4 in binary (i.e. PCU3 is represented by the token "011".)

The DDL also used to describe the file more generally. For event modes, the TDDES2 keyword uses the DDL to describe what type of events may be present in that particular file; in other words, it tells from which PCU, anode, and channel the events may be from. It does not necessarily mean you may select on these values, however, as that token itself may not be present in the event word. For instance, a common mode (E_16us_64M_36_1s_fits) has

```
TDDES2  = 'D[0~4] & E[X1L^X1R^X2L^X2R^X3L^X3R] & C[36~254]'
```

which tells you three things:

1. D[0~4] tells you there will be events from all detectors present
2. E[X1L^X1R^X2L^X2R^X3L^X3R] tells you that there will be events from all layers, but no coincidence events ('^' is the logical EOR, or 'exclusive or'; "X1L", etc. is special case where the anode ID is used instead of the value.)
3. C[36~254] means that events from PHA channels 36 and above will be present.

It is the TEVTB2 keyword which tells you what you can actually select on and how:

```
TEVTB2  = '(M[1]{1},C[ ... ,42,43,44~45,46~47 ... ]{6},T[ ... ]{9})
```

where the " ... " is the long, detailed description of the channel binning or the timing accuracy (see below.) This tells you the only tokens actually used in the event words. There is no D token, for instance, so you cannot tell which detector an event came from. Nor can you tell which anode. In this example, too many bits are taken up with the timing and channels for other information to be included.

In the square brackets is the information about what values the token may take. For instance, the C token contains the channel binning information. When there are fully 256 channels represented in the event word, this token will appear as C[0:255]{8}, indicating that it takes up 8 bits to represent any value from 0 to 255. If the channels are binned to fewer than 256, then that binning information will appear within the brackets. For example, in a binned mode with only two channel bins, the TDDES might appear C[0~13,14~49], indicating that absolute channels 0 to 13 are in the first bin and 14 to 49 in the second. A more complicated token, like the 64M binning above, may take several lines in the TEVTB keyword to tell exactly which absolute channels are binned into which relative channels. **[*Note: the information contained in the C token is meant to be used by the XTE software only and is described here simply as an example of the DDL. Do not attempt to filter your data by channel using this token. All channel filtering should be done in* seextrct *in absolute channels (see the ABC Guide's page about the extractors).*]**

- ## Determining what you can filter:

The only two things you should use a bitmask to filter on are the D (detector) token and the E (anode) token, or sometimes the Z (combined detector and anode) token. For example, the E[0:63]{6} token in the TEVTB2 keyword stands for E[X3R,X3L,X2R,X2L,X1R,X1L], i.e. each bit represents an anode. The pattern "010000" represents an event in X3L, while the pattern "110000" represents a patter which triggered both X3R and X3L. So if you want to select layer two, non-coincidence events only, you want the values 4 or 8. A six-fold coincidence event, on the other hand, would have the value 63.

If a token is not present in the TEVTB2 keyword, then that means the event words do not have that information in them and you can't filter on it. You can, however, see from the TDDES2 keyword which values of that token will be included in the event list. For instance, a common mode (E_62us_64M_0_1s_L1R1) has

```
TEVTB2  = '(M[1]{1},D[0:4]{3},C[ ... ]{6}T[ ... ]{6} ... "
```

which means that you can filter on the D token only, not the E. Above that in the header, you'll see

```
TDDES2  = 'D[0~4] & E[X1L^X1R] & C[0~249]'
```

which means that there are events from detectors 0-4, layer one, and channels 0-249 present in the file. So you can filter out the detectors you want, but you will only get layer one and nothing from absolute channels 250-255.

Using sefilter

sefilter is a Perl script which reads the data descriptor for you, determines which tokens you may filter on, and queries you for the values you want. It is not so smart, however, to ask something like "what detectors do you want" or "which layers do you want"; it only understands DDL, so you have to be familiar with the actual bit values and tell it what you want in terms of those values.

A few things to note about sefilter and event modes:

1. All event words containing science data - as opposed to housekeeping data - are prefaced with the M-token `M[1]{1}` which tells whether an event is a time stamp or a real event. (Time stamps are present in all science event data with the exception of GoodXenon mode.) Selecting this token is the default in sefilter, and even if no other selection is desired, these time stamps must be filtered out before analysis.

2. Not selecting on a given token is equivalent to selecting all possible values.

3. Sefilter accepts input in denary not binary; it will translate into the right expression if you simply give the correct denary value. For instance, "2" should be entered for the D[0:4]{3} token to select PCU2, not "010" which is the actual token.

4. Seextrct cannot handle all valid bitmasks, in specific any bitmask containing a logical OR. If you have constructed one of these with sefilter, then sefilter will simply apply this bitmask with fselect to produce a bitmask-filtered data file which can be input subsequently into seextrct. If, on the other hand, your bitmask can be handled by seextrct, sefilter will give you the choice of proceeding directly to seextrct or of producing a bitmask-filtered data file with fselect.

5. This tool will take your input values and construct a logical expression to represent them. It will display this before moving on to the next token. (At the end, it will give you the option of storing the combined expression for your reference.) You should note that if you enter two adjacent values separated by a comma, e.g. "1,2" for detectors, it will construct the expression using an OR (D[0:4] == 1 | D[0:4] == 2) which cannot be handled by the extractor. Whenever possible, use a dash, e.g. "1-2", which will result in an AND expression (D[0:4] >= 1 & D[0:4] <= 2).

As you can tell from the above, PCU and anode selection for science event data is not so simple that one fully prescriptive recipe will work for all cases. Accordingly, here are two examples, one for the Good Xenon configuration (which as D-tokens and E-tokens) and one for a generic event configuration (E_1ms_128M_0_8s) which has Z-tokens. For a start-to-finish example of an sefilter run, see the ABC Guide's page about extracting event data.

GoodXenon example:

- At the risk of stating the obvious, we reminded you that GoodXenon data come to you in two separate files containing the two halves of the extra long event word. These must be merged to form the FITS event list using the script **make_se** before any other step. This will produce one FITS file for each corresponding pair of incomplete files you put in.

- This mode generally has three tokens to filter on: the D[0:4]{3} and the E[0:63]{6} are the familiar detector and anode tokens, but recently some observers have requested the option of examining the propane layer events, so a new token was added, the E[VPR]{1} which will come first. These events will only be present in your data if you specially request them, but the token will be there for any data processed by the latest v4.0 xenon2fits (as called by make_se); for almost all observations, this token will be zero for every event, so you may ignore it. If you have requested propane events, you will want to separate these events from the others by making one file with E[VPR]==True and another with E[VPR]==False. (Obviously, if E[VPR] is True, then the E[0:63] token will be zero.)

- After the E[VPR] token comes the D[0:4] token, which is the usual detector token with the obvious values.

- Next is the E[0:63]{6} token. If you look in the header of the GoodXenon FITS file, it has a long COMMENT explaining the values for the different anodes. In sefilter, you will simply see:

  ```
  This file contains an E-token, specifically:
  E[0:63].
  We have 6 bits for this token.
  So this token can take on values from 0 --> 63

  E-token values from 0 to 63 are present.
  Do you want to filter on this E-token? [Yes, No]: yes

  Enter the layer values of interest: [0-10, 15, 22-50]
  The above syntax must be followed:
  ```

  where you are asked for the values of the E token which represent the anodes you want. For layer one, right anode, the token would look like "000010", which has the value of 2, etc. So, if you wanted to filter out layer one, you would answer the prompt with "1-2". This will construct a bitmask using a logical AND, i.e. the expression passed to sebitmask will be:

  ```
  E[0:63] >= 1 & E[0:63] <= 2    .
  ```

- So, if you did not choose to filter on detector, but did choose layer one, you would end up with the following bitmask:

```
Event >= bxxxxxxxxxx000001xxxxxxxx &&
Event <= bxxxxxxxxxx000010xxxxxxxx &&
Event == b1xxxxxxxxxxxxxxxxxxxxxxx
```

The last line is the default filtering of the M[1]{1} token, which removes the time stamps which are present in all event modes except GoodXenon, where that bit is always 1. If you delete that line, sefilter may run somewhat faster, although for the extractor it should make little difference.

- If, however, you wanted detectors 0-2, and 4, then the bitmask would include a logical OR corresponding to the expression D[0:4] <= 2 | D[0:4] == 4 :

```
Event == b1xxxxxxxxxxxxxxxxxxxxxxx &&
(Event <= bxxxxxxx010xxxxxxxxxxxxxx ||
Event == bxxxxxxx100xxxxxxxxxxxxxx )
```

and you would not be given the option of bypassing fselect and going straight to the extractor.

Generic Event example:

In event-mode configurations with both IDs, *either* the PCU ID is encoded in the D-token and the anode ID is encoded in the E-token, as for GoodXenon, *or* the PCU and anode ID are both encoded in the Z-token. Of course, if the configuration doesn't have anode ID, only the D-token is present. And if the configuration doesn't have PCU ID, neither token is present.

- If your configuration has the Z token, then you should see a table of values corresponding to the various combinations of PCU and anode. They are self-explanatory:

```
Beginning generation of filter description.
###########################################

This file contains a Z-token, specifically:
Z[D[0]&E[X1L],D[0]&E[X1R],D[0]&E[X2L],D[0]&E[X2R],D[0]&E[X3L],
D[0]&E[X3R],D[1]&E[X1L],D[1]&E[X1R],D[1]&E[X2L],D[1]&E[X2R],D[
1]&E[X3L],D[1]&E[X3R],D[2]&E[X1L],D[2]&E[X1R],D[2]&E[X2L],D[2]
&E[X2R],D[2]&E[X3L],D[2]&E[X3R],D[3]&E[X1L],D[3]&E[X1R],D[3]&E
[X2L],D[3]&E[X2R],D[3]&E[X3L],D[3]&E[X3R],D[4]&E[X1L],D[4]&E[X
1R],D[4]&E[X2L],D[4]&E[X2R],D[4]&E[X3L],D[4]&E[X3R]].
We have 5 bits for this token.
So this token can take on values from 0 --> 31

The Z-token is SPECIAL. So we will deal with it in detail.
Number of Z-tokens is 30
D[0]&E[X1L] ==  0          D[2]&E[X2R] == 15
D[0]&E[X1R] ==  1          D[2]&E[X3L] == 16
D[0]&E[X2L] ==  2          D[2]&E[X3R] == 17
D[0]&E[X2R] ==  3          D[3]&E[X1L] == 18
D[0]&E[X3L] ==  4          D[3]&E[X1R] == 19
D[0]&E[X3R] ==  5          D[3]&E[X2L] == 20
D[1]&E[X1L] ==  6          D[3]&E[X2R] == 21
D[1]&E[X1R] ==  7          D[3]&E[X3L] == 22
D[1]&E[X2L] ==  8          D[3]&E[X3R] == 23
D[1]&E[X2R] ==  9          D[4]&E[X1L] == 24
```

```
D[1]&E[X3L] == 10          D[4]&E[X1R] == 25
D[1]&E[X3R] == 11          D[4]&E[X2L] == 26
D[2]&E[X1L] == 12          D[4]&E[X2R] == 27
D[2]&E[X1R] == 13          D[4]&E[X3L] == 28
D[2]&E[X2L] == 14          D[4]&E[X3R] == 29
Do you want to select any of these? [Yes, No]:
```

So for example, to select the top layer (`E[X1L]` & `E[X1R]`) of PCU0-PCU3, (`D[0]` - `D[3]`) then specify: "0-1,6-7,12-13,18-19" to get the bitmask:

```
Event == b1xxxxxxxxxxxxxxx &&
(Event <= bxxxxxxxx00001xxx ||
Event >= bxxxxxxxx00110xxx &&
Event <= bxxxxxxxx00111xxx ||
Event == bxxxxxxxx01100xxx ||
Event >= bxxxxxxxx10010xxx &&
Event <= bxxxxxxxx10011xxx )
```

- If your configuration has no detector or anode information, then at this point, the line "Beginning generation of filter description." will be followed by nothing and the only expression used will be the default filtering of the M[1]{1} token to remove the time stamps.

At this point, sefilter will examine your selections and tell you whether the corresponding bitmask can be entered into *either* seextrct and fselect *or* fselect only. If you do have the choice between fselect and seextrct, then consider that:

- **Fselect** will produce another science event file containing only your selected events. As such, it can be entered into seextrct without further bitmask-selections (a convenience) but is likely to be large (an inconvenience).

- **Seextrct** will directly apply the bitmask when it extracts your light curve and/or spectrum. However, if you want to do a second extraction with the same PCU and anode ID selections, you must remember to enter the bitmask on the command line with, e.g. `seextrct bitfile="E_1ms_128M_0_8s_pcu0_top.bit"` (an inconvenience). But you won't need to store an intermediate science event file (a convenience).

After making the choice between fselect and seextrct, you'll see the question:

```
Expression generated by SEFILTER stored in outfile_2587
Do you want to save this file? [Yes,No]
```

We strongly recommend saving this file and giving it the same name has the bitmask but with the extension .bitsel. In this way, you be able to look at, say, E_1ms_128M_0_8s_pcu0_top.bitsel and verify what the corresponding bitmask E_1ms_128M_0_8s_pcu0_top.bit contains.

- ## A note on strategy:

You can use the information in both the TEVTB2 and TDDES2 keywords to figure out the best way to filter your data. If the TEVTB2 says E[0:63]{6} and the TDDES2 says E[X1L^X1R^X2L^X2R^X3L^X3R], the anode information is there but only non-coincidence events are included in the file, so no values but powers of two would appear in the file. To ask for E[0:63]==24 in sefilter would be perfectly valid, but would result in an empty file. The advantage of this is in simplicity.

Say you wanted layers one and two, corresponding to values 1,2,4, and 8. If you have a file with coincidence events, you will have to specify these values by entering "1,2,4,8" at the sefilter prompt. It will then form an OR expression, which cannot be used by seextrct. If, however, the file contains no intermediate values, you can simply say "1-8", forming a simple AND expression, and you'll get the right result in seextrct, bypassing the lengthy fselect process.

XTE Project Scientist Jean Swank, swank@pcasun1.gsfc.nasa.gov
Technical Rep: Sherri Calvo, sherri.calvo@gsfc.nasa.gov
