IDL Extract:
A System for Analyzing RXTE, Chandra and Swift Data in IDL

Craig Markwardt, GSFC
Craig.Markwardt@nasa.gov

Abstract

This document describes the analysis of *Rossi X-ray Timing Explorer* (*RXTE*) Proportional Counter Array data using a suite of routines in IDL. The tools can also be used for Swift XRT and Chandra event data. The high level tools can make light curves, spectra and Fourier power spectra, all within the programmatic environment of IDL. The programs allow particularly easy ways to filter and bin data in time, energy and frequency, with special attention to consistency across all programs. The output files are written in standard FITS formats where appropriate, which are compatible with standard downstream software. This library specializes using RXTE data to its fullest, including making time-resolved energy and power spectra, but can do more simple operations such as light curves and spectra.

1 Introduction

This document describes a package of routines called "IDL Extract," which aid in analyzing data from *RXTE*. You will find that IDL Extract can supplement — and even extend — the realm of possibilities defined by FTOOLS, but IDL Extract cannot directly replace FTOOLS itself. IDL Extract is also able to accept Chandra and Swift XRT event data.

To most scientists, IDL Extract provides a way to extract power spectra, light curves, and energy spectra from *RXTE* data. At a lower level there are routines that can assist in directly loading data from raw *RXTE* files, but I anticipate that most people will want to:

- **Make light curves** — IDL Extract will make light curves from *RXTE* data files, and optionally output OGIP-compliant FITS files. An option is provided to extract data on a per-detector or per-layer basis. (see `BINLC`)
- **Make energy spectra** — IDL Extract will extract single PHA spectra from input files, and again optionally output to a PHA FITS file suitable for use with XSPEC. (see `BINPHA`)
- **Make "dynamical" power spectra** — IDL Extract can construct multiple Fourier power density spectra from an *RXTE* input file, producing a "sonogram" of the source's spectral power (see `RADPS`). Optionally, `RADPS` will average the spectra over any desired interval or rebin in frequency.
- **Make "dynamical" energy spectra** — IDL Extract can perform the same analysis in the energy domain, and optionally create a "Type II" PHA file. These files are ideal for loading many spectra into XSPEC. (see `RADES`)

- **Construct the cross spectrum** — IDL Extract will generate the phase lag and coherence spectrum between two energy bands of an *RXTE* data set.

In addition to these capabilities, IDL Extract has several other benefits to you as a scientist.

- It will intelligently handle almost all PCA data modes, including the event modes. IDL Extractor automatically constructs the proper event "bit-mask" depending on which filters you apply.
- It is often faster than the SAEXTRCT/SEEXTRCT equivalent.
- It permits you to combine *RXTE* data sets which are parallel in time. For example, if the data you wish to analyze contains simultaneous single-bit and event mode data sets, you can combine those files directly in a single call to any one of the analysis routines. IDL Extract will intelligently stitch them together.
- Analysis in IDL is generally more flexible, because you can directly manipulate the data as arrays once the output has been extracted.
- It understands most "standard modes" such as event, single-bit, binned, event, and good xenon.

There are some drawbacks however. First and foremost, because IDL is a very flexible environment, you will find that you have to keep track of variables and data arrays yourself. Some people may find that they need to do very simple analyses. In that case, I recommend that they stick to the FTOOLS because those are well supported and fairly complete.

Also, IDL Extract is not meant as a complete replacement for the FTOOLS. There are some portions of FTOOLS which perform highly customized tasks, such as background estimation, and spectral fitting, which cannot be replicated simply in IDL. When you need to perform background estimation, etc., you will have to return to the FTOOLS. More aptly, IDL Extract is meant to supplement the following FTOOLS: SAEXTRCT, SEEXTRCT, SBMERGE, SEMERGE, POWSPEC,

In other respects there are some disadvantages:

- IDL Extract does not have a graphical user interface.
- It is difficult to call IDL Extract from the Unix command line.

In the next few sections I describe how IDL Extract can be used to perform basic tasks. First I cover constructing a binned light curve, followed by simple power spectral analyses. Then I proceed to examine more advanced procedures, such as dynamical energy and power spectra. I probably cannot cover everything in this documentation. Despite that, I have tried to make the behavior of each program as consistent as possible with all of the others. Hopefully you can take what you have learned about IDL Extract and transfer it to other tasks.

There are other information resources. You should also be aware that the source code of each of the main programs contains extensive "reference" documentation in their header,

which describes how to use the programs in more detail. That material is included at the end of this document as well. Finally, one of the best ways to understand how something works is to read the source code itself. The main programs are well-documented throughout, and may provide answers to questions. Thankfully, it is fairly easy to read the IDL language.

2 Getting Started

Please see the Appendix section on installation of the software.

To test that things are working properly, enter IDL and enter this command:

```
IDL> binlc
% Compiled module: BINLC.
% BINLC: BINLC, [INFO1,INFO2,...], LC, LCINFO,
% BINLC:        TBINSIZE=, ...
IDL>
```

`BINLC` provides a short usage message. If you receive compilation errors, then you should check to be sure that your path is set correctly and that the IDL Extract directories fully populated.

3 Making a light curve with BINLC

At its most fundamental, making a light curve with IDL Extract involves choosing a time resolution and running `BINLC`. All of the IDL Extract commands have additional options which allow you to refine data selection, time ranges, and so on.

Let us say that we have a binned-mode data file, and wish to make a one-second light curve. Here is the command:

```
IDL> filename = 'FS37_4153340-415408a'
IDL> binlc, filename, lc, tbinsize=1.D, time=tt
*********************************************
BINLC: Bin a light curve
 Time range:    68498240.000-   68501640.000 s  (MET)
 Filename: FS37_4153340-415408a
     Event PHA Channels : 0 to 249      (file bins 0 to 63)
---------------------------------------------
    Time binsize:        1.0000000 s    (      0.25000000 Hz)
  Number of bins:         3400
       Channels:       0-249
  Total duration:        3400.0000 s
     Buffer size:        8.0000000 s
   Detector list: 0,1,2,3,4, merged
      Layer list: X1L,X1R,X2L,X2R,X3L,X3R, merged
---------------------------------------------
```

BINLC is the generic light curve generator. You supply it with the file names that contain the raw data, and then it bins the light curve for you, returning it in the variable lc. The "time" variable is returned in TT. You specify the time resolution of the light curve using the TBINSIZE keyword option.[1] In this example, lc is an array of count rates.

Once you have the light curve, it can be plotted. Since we have extracted a light curve with one second time bins, you can plot it directly using the IDL PLOT procedure:

```
IDL> plot, lc
```

When you do not supply an "X" value to PLOT, then it separates each point by 1 unit, which is perfect for this particular application, which has one second bins. In the case where the time binning is important, use the TT variable, like this,

```
IDL> plot, tt, lc
```

Clearly, you will want to extract light curves with different bin sizes. See the following sections for more information.

3.1 The status line

After entering the command above, BINLC began by printing a short message describing the operation, and then proceded to accumulated the light curve. As it continues, BINLC will print and update a one-line status message like the following:

```
 RE  Rows   655361-  671744          5  File FS37_4153340-415408a         3%
----        ---------------      -----       --------------------       ----
Current      Current Row       Current        Current File Name     Percentage
Operation      Number     FFT or Spectrum                             Complete
```

You should keep an eye on the percentage complete, which is updated periodically, to know how long the job will take. Since the status line accesses the console directly, none of it should appear in your journal file.

3.2 Getting the error bars and the time

No scientific study is complete without error analysis. You can, by specifying further keyword options to BINLC, retreive error bars for each light curve time bin. BINLC assumes Poissonian errors in the gaussian limit. More simply put, it takes the square root of the number of counts in the bin. You of course may do this by yourself if you wish; IDL Extract does make it easier by computing the error in the *rate* rather than total counts. Here is the command:

```
IDL> binlc, filename, lc, tbinsize=1.D, error=elc, time=tt
```

Since we have passed a variable called ELC using the ERROR keyword, BINLC will return the statistical error in that variable.

[1] There are other ways of specifying the time resolution. See the keywords SAMPFREQ, NYQUISTFREQ, and (for Fourier applications) NFBINS for the program you are interested in.

As noted above, the other keyword, `TIME`, instructs `BINLC` to return the time of each bin in the variable `TT`. Upon return, `TT` will contain the "Mission Elapsed" time (MET), in seconds, of the *beginning* of the time bin.[2] Using this information, you can now plot a full light curve with errors in IDL:

```
IDL> ploterr, tt, lc, elc
```

where `PLOTERR` is a standard IDL routine which plots data and errors. Although the time variable is returned with double precision, the IDL plotting routines only keep single precision. If you see peculiar quantization or "staircase" effects, then it may be wise to plot with a time offset. This is probably good practice anyway, and is easy to do. Simply subtract the first time value:

```
IDL> ploterr, tt-tt(0), lc, elc
```

3.3 Time Filtering: Explicit Time Limits

You may find that you wish to limit the time range of consideration in your light curve. By default, `BINLC` will generate a light curve for the entire good time range of the input file(s). This is accomplished very simply by passing the starting and stopping time (in MET seconds) in a vector to the `TLIMITS` keyword:

```
IDL> binlc, filename, lc, tbinsize=1.D, tlimits=[68498240.D, 68498440.D]
```

This command limits the light curve to a smaller 200 second window at the beginning of the input file. Remember to *always* specify the time in double precision. A single-precision variable is not sufficient to store large *RXTE* mission times. The "D" in the numbers above indicate double precision.

3.4 Time Filtering: Good Time Intervals

Sometimes you will have a more complicated set of time intervals. This is most naturally expressed as a set of Good Time Intervals (or GTIs). When using IDL Extract, a GTI is simply a $2 \times N$ array of start and stop times, expressed in MET. For example, if you were interested in the first 200 second and last 200 seconds of the data file, you could make a GTI array like this:

```
IDL> my_gti = 68498240.D + [[0,200], [3200,3400]]
```

This array has a 200 second interval at the beginning and a 200 second interval at the end. Here I have taken advantage of the fact that IDL will automatically add the constant scalar value to each element of an array. This value (MET = 68498240) is the start time of the observation.

You apply this time filter using the GTI keyword, such as the following command:

```
IDL> binlc, filename, lc, tbinsize=1.D, gti=my_gti
```

[2] You can adjust this by using the `TPIXR` keyword.

It is also possible that you have stored the good time intervals in a separate file on disk using the standard GTI format. Often times this can be done with the FTOOLS program called 'maketime'. As a special case, the GTI keyword can be a string, and BINLC will automatically read the contents of the file. For example, if the good time intervals are stored in a file called '`good_data.gti`', then you could use the following invokation to apply time filtering:

```
IDL> binlc, filename, lc, tbinsize=1.D, gti='good_data.gti'
```

More sophisticated time filtering can be done with the GTIREAD and GTIMERGE functions. You can also create good time intervals based on any filtering criteria using the MAKETIME IDL procedure.

3.5 Filtering by Energy

If your data has channel information, then you can also select counts based on their energy. Data modes such as Standard2, Binned and Event can have pulse-height channel information. Given that quite frequently that observations are background-dominated above a certain energy level, it may sometimes make sense to limit the channel range of your light curve.[3] The keyword to use is `CLIMITS`:

```
IDL> binlc, ib, lc, ilc, tbinsize=1.D, climits=[0,81]
```

In this case only energy channels 0 to 81 inclusive are used in constructing the light curve. Please note that you should specify the pulse height *channels* (which always range from 0–249), and not a mode-specific *binning.* Regardless of the observing mode, channel assignments are always the same.

3.6 Combining multiple input files

IDL Extract is very good at combining data files that come from different data modes or different times. You are allowed to specify as many input files as you wish at the input command line.

Consider an example. Observations of some sources may be composed of Single-Bit and Event, or Binned and Event. IDL Extract will combine those two modes automatically. With FTOOLS you must first generate separate light curves for *each* observing mode and then add them together using "lcmath." With IDL Extract, you simply supply the addition INFO variables to `BINLC` directly, as in the following:

```
IDL> binlc, [binned_file, event_file], lc, tbinsize=1.D, climits=[0,81]
```

where `binned_file` and `event_file` are variables which contain the names of the two files to be processed. You must pass both files together, as an array, and `BINLC` will understand to combine them into one output. There are of course some restrictions:

There are two important requirements: each data file must not overlap with any other data file in both time and energy. This means that you can specify two files that cover two

[3]For a conversion chart between PHA channel and energy, see the *RXTE* GOF web page.

different energy ranges at the same time, or two files that cover two different time ranges in the same energy band, but not both. Another requirement is that if you specify files that cover different energy ranges, then you must specify coverage for those energy bands for all times you are interested in.

IDL Extract can in principle combine many modes into one, but the most is three in practice, I think.

3.7 Saving output to a file

By default, IDL Extract programs will generate an in-memory array of data. You may manipulate this array directly if you wish. Sometimes however you may wish to save the output for later access. You can accomplish this in two ways: (1) you can use the IDL `SAVE` procedure, which saves all variables in the proprietary IDL format; or (2) use the `OUTFILENAME` keyword. If you pass a file name (as a string) using the `OUTFILENAME` keyword, then `BINLC` will save the output in an OGIP-standard FITS light curve file. Here is how it works:

```
IDL> binlc, filename, lc, tbinsize=1.D, outfile='burst01.lc'
```

Data is written to the first extension of the output file, and a GTI (good time interval) array is written to the second extension. You can use the FTOOLS to read the light curve and perform subsequent analysis if you wish. IDL Extract does not directly read product FITS files such as light curves, but you can do this yourself by using the BINTABLE routines (which IDL Extract itself uses). Here is an example of re-reading an FITS light curve file:

```
IDL> fxbopen, lcunit, 'burst01.lc', 1
IDL> fxbread, lcunit, t, 'Time'
IDL> fxbread, lcunit, r, 'Rate'
IDL> fxbclose, lcunit
```

3.8 Accumulating light curves for individual detectors

Some data files contain individualized detector and/or layer information. Keeping this information distinct may be especially useful if you need to know whether a particular detector is turned on, or if you wish to accumulate a top-layer light curve for a faint source.[4] The default is for `BINLC` to *combine* this information into one light curve series. You may request that `BINLC` not perform this action by setting either the `NOMERGEDETS` or the `NOMERGELAYERS` keywords:

```
IDL> binlc, filename, lc, tbinsize=1.D, /nomergedets
```

As a result, the light curve is actually an *array* of values: one vector for each detector/layer combination requested.[5] If saved to a FITS output file, then multiple light curve

[4] The signal to noise ratio is higher in the top layer.

[5] One additional caveat applies: you must specify `NOMERGEDETS` if you want `NOMERGELAYERS`. That is, if you want individual layer information then you must also request individual detector information. If you wish, you can combine the detector information afterward.

columns appear, each with its own appropriate column name. Of course, if the data does not have detector or layer information, then a warning message will appear and the light curve cannot be generated.

4 Making a Power Spectrum with RADPS

Now let's move on to the next topic: making a Fourier power spectrum using RADPS. Like most timing spectral analyses, RADPS uses the Fast Fourier Transform (or FFT) in order to estimate the power at each frequency. If you are used to timing software like XRONOS, then you may be expecting to create a light curve first and then compute the power spectrum from that. However, that is *not* how RADPS works. In fact, RADPS does not allow you to use a light curve as input. Instead, you provide the raw data, and RADPS creates the necessary light curve internally, and the power spectrum is computed from that. This is usually what you want, since the intermediate light curve is rarely needed for anything else, and it often occupies a lot of disk space.

If you are familiar with timing analysis, then you should know that it is customary to break a given data set into different time segments, estimate the power spectrum from each segment, and then average the power spectra. This process is needed in order to average away the large variations inherent in the powers of a single power spectrum. RADPS is capable of doing this, as well as several other options.

The basic command is invoked like this:

```
IDL> ffttime = 128.D
IDL> RADPS, filename, ffttime, dps, nyquistfreq=1024.D, avgtime=-1
```

where `FILENAME` is the name of the input file and `FFTTIME` is the duration of a single FFT (here set to 128 seconds). The `NYQUISTFREQ` setting is the Nyquist sampling frequency in Hz, and specifies the maximum possible frequency of the power spectrum (you can also specify `TBINSIZE` or `SAMPFREQ` instead of `NYQUISTFREQ`). The smallest possible frequency is given by 1/`FFTTIME`. The `AVGTIME` setting tells RADPS to average all power spectra together into a single output spectrum. The output will be stored in the variable named `DPS`.

If you run this command, it will produce a single power spectrum, which will be stored as a 1-dimensional vector in `DPS`. It is possible to plot this spectrum with the command:

```
IDL> plot, dps
```

However, like the light curve example above, the X-axis of the plot is arbitrary. To get an appropriate frequency axis, you should re-run RADPS with the `FREQAVG` `FREQWID` keywords, and RADPS will provide the frequency labels and widths for each power spectrum sample. For example,

```
IDL> RADPS, filename, ..., freqavg=ff, freqwid=fw
```

Then it will be possible to make a proper plot,

```
IDL> plot, ff, dps
```

The `FW` array gives the width of each bin, and should be equal to 1/`FFTTIME` for this example.

4.1 Filtering Power Spectra by Time or Energy

The same keywords — `TLIMITS`, `GTI` and `CLIMITS` — are used by RADPS in order to filter the input data by time and energy. You can read more about these in the section above about BINLC.

4.2 Making Dynamical Power Spectra

A dynamical power spectrum here refers to calculating a power spectrum for multiple time intervals. This kind of analysis will be important when it is suspected that the timing properties of the astrophysical object might change rapidly. The desire to extract dynamical power spectra should be balanced with the understanding that individual, unaveraged power spectra are quite noisy, so any sought-after spectral features must be quite significant.

To make a dynamical power spectrum, simply remove the "AVGTIME=-1" keyword, from the command line, such as this,

```
IDL> RADPS, filename, ffttime, dps, nyquistfreq=1024.D
```

Without the `AVGTIME` keyword, RADPS will preserve each spectrum that it computes, and they will all be returned in the `DPS` variable. The dimensions of the variable will be $N_{\mathrm{fbins}} \times N_{\mathrm{tbins}}$ where N_{fbins} is the number of frequency bins, and N_{tbins} is the number of time bins. The number of time bins will be approximately equal to the total observation duration divided by `FFTTIME`. However, because it is possible to have fractional coverages (portions of the light curve that do not have an exact duration of `FFTTIME`), some spectra may be rejected. This is also controlled by the `MINFRACEXP` keyword.

Just as for BINLC, the `TIME` keyword can be used to extract the time label of each power spectrum.

Please be aware that it is possible to overflow your computer's memory if too many spectra are requested, especially if the number of frequency bins and the number of time samples is large.

4.3 Dynamical Power Spectra with Some Time-Averaging

The previous sections described ways to make power spectra where *all* data are averaged, and *none* of the data are averaged. There are some intermediate possibilities as well.

You may use the `AVGTIME` keyword to specify an any averaging interval you wish (which is a multiple of `FFTTIME`). For example, if FFTTIME=128 and AVGTIME=256, then two adjacent spectra will be averaged into one output sample.

The `NOCROSSGTI` gives you another way to control how power spectra are averaged. The most common use will be AVGTIME=-1, NOCROSSGTI=1, which indicates to average all power spectra within a contiguous good time interval. This is usually a quite useful setting that compute the mean power spectrum per observation.

Note that the number of averaged spectra is returned with the `NSPECSUM` keyword. The value may be different for each spectrum, especially if `NOCROSSGTI` is used, so the quantity returned is a 1-dimensional vector, with one sample for each averaged power spectrum.

4.4 Normalization

It is important to know the normalization of the power spectrum, since the normalization determines the properties of source and noise.

The normalization returned by RADPS is the same as Leahy (1983),

$$P_j = \frac{2}{N_{\rm ph}} |a_j|^2,$$

where $N_{\rm ph}$ is the number of counts in the spectrum, and a_j are the Fourier coefficients from the power spectrum. The important property of this normalization is that the power spectrum of pure Poisson noise has a chi-square statistical distribution with a mean value of 2 and a standard deviation of 2 (for 2×`NSPECSUM` degrees of freedom). Thus, this is the best normalization to use when performing a test for a significant sygnal, since the test reduces to a basic chi-square test. (For example, given a measured Leahy-normalized power of 4.2 for one bin in one power spectrum, the significance would be 12%).

A second well-known normalization is given in units of (r.m.s./mean)^2/Hz, and specifies the fractional variance per frequency interval. This normalization is most useful when computing the variability of a source over a given frequency range (after subtracting the noise level of course!). You must compute this normalization by hand, as follows,

```
IDL> RMS_MEAN_HZ = (DPS/MEAN_RATE)
```

where `MEAN_RATE` is the mean rate obtained from the `PO` keyword as described above.

Now let us say we wish to compute the fractional r.m.s. variability between 10 and 20 Hz. First, we select the desired frequencies between 10 and 20 Hz like this,

```
IDL> WH = WHERE(FF GE 10 AND FF LE 20)
```

and second, we compute the fractional variability by integrating over the desired frequency range and taking the square root,

```
IDL> RMS = SQRT( TOTAL( RMS_MEAN_HZ(WH) * FW(WH) ) )
```

Here `FF` and `FW` are the center frequency and frequency width as reported by the RADPS keywords `FREQAVG` and `FREQWID`. The final result will be in unitless, and express the fractional variation of the source (you can multiply to 100 to obtain a percentage).

5 Making an Energy Spectrum with BINPHA

Now let's move on to making energy spectra. The basic procedure for doing this is BINPHA.

By default, BINPHA will make one spectrum from all data. Invoke it like this,

```
IDL> binpha, filename, pha
```

where `FILENAME` is the input file name, and `PHA` is the name of a variable which will be assigned the output spectrum. By default, the pulse height bins in the input file are preserved without rebinning.

Just as for BINLC, the uncertainty in the spectrum can be retrieved with the ERROR keyword.

For plotting purposes, it is nice to retrieve the channel label for each bin, and as well as the width of each bin. By direct analogy with the `FREQAVG` and `FREQWID` keywords, the `CHANAVG` and `CHANWID` keywords can be used for this purpose. A nice plot can be made like this, including error bars,

```
IDL> plot, chanavg, pha/chanwid, psym=10
IDL> oploterr, chanavg, pha/chanwid, error/chanwid
```

5.1 Rebinning by Energy

As already mentioned, the output spectra has the same binning as the original raw data file. You can request the file to be rebinned by using the CBINS keyword. This keyword should contain a $2 \times N$ array giving the edges of each desired bin. For example, to make a 4-bin spectrum containing the channel ranges 5–9, 10–29, 30–79, 80–249, use the following array:

```
IDL> cbins = [[5,9], [10,29], [30,79], [80,249]]
IDL> binpha, ..., cbins=cbins
```

Please be aware that the requested energy bins must not conflict with the energy bins of the original input file. You can use the XTEINFO procedure to list the binning of the input file.

5.2 Filtering Energy Spectra by Time

The same keywords — `TLIMITS` and `GTI` — are used by BINPHA in order to filter the input data by time. You can read more about these in the section above about BINLC.

By default all the selected times are combined into a single output spectrum. However, you can request that the input data be broken into one spectrum for each good time interval, using then NOCROSSGTI=1 option. In that case, the PHA variable will contain a two dimensional array with the number of bins on one axis, and the number of time samples on the other axis. If you want finer control over the time sampling, use RADES instead.

6 Making Time-Resolved Energy Spectra with RADES

RADES is specifically designed for making time-resolved spectra. Just as for RADPS, the input data are divided into equal-sized time intervals, and one spectrum is computed for each interval.

The basic command is written like this,

```
IDL> SPECTIME = 128.D
IDL> rades, filename, spectime, DES, avgtime=-1
```

where `SPECTIME` is the requested exposure of each spectrum, and AVGTIME=-1 indicates to perform no averaging.

By analogy with BINPHA, the energy binning can be controlled with the CLIMITS and CBINS keywords.

By analogy with RADPS, the time sampling can be controlled with the AVGTIME, GTI and TLIMITS keywords (as well as MINFRACEXP).

7 Advanced Usage: Quality Checking

Every user is responsible for checking the quality of the processing and the output data. In addition to the kinds of things that only a human can do, the IDL Extractor tools provide some standard quality filtering information.

The `STATUS` keyword is filled with a value of 1 if the processing succeeded, or 0 if it failed. Users should check for serious failures using the results of this keyword.

Also, the `QUALITY` keyword will contain a measure of the quality of each time and/or energy sample. This is especially useful if some time samples or energy samples were flagged as "bad" in the input file, but most of the data was OK. "Good" output samples are indicated by a quality value of 0. Any other quality value indicates "bad" output data.

8 Writing Results to an Output File

All IDL Extract programs support writing their results to an output file. When the `OUTFILENAME` keyword is specified, then a file is created with the following format:

BINLC	Standard OGIP-format light curve
BINPHA	Standard OGIP Type I or II spectrum
RADES	Standard OGIP Type II spectrum
RADPS	Custom-format spectrum

In the case of BINPHA, the output is either in "Type I" or "Type II" spectrum format, depending on whether multiple spectra were requested, and also on whether the `OUTTYPE` keyword is set to 'TYPE_I' or 'TYPE_II'. Either format is recognized by standard spectral fitting packages like XSPEC.

Since there is no standard format for power spectral analysis, the output of RADPS is a custom-format FITS file.

9 Appendix: Standard Keywords

This appendix describes the standard keywords that are applicable to all IDL Extractor procedures. They are described as Input or Output depending on whether the argument is expected upon input, or will be returned upon output.

Here is a table which may help you decide which keywords to use in order to filter or resample your data by time, energy, frequency or detector.

If you want to...	... filter by ...	... bin by ...
... time, use ...	TLIMITS, GTI	TBINSIZE, SAMPFREQ, NYQUISTFREQ
... energy, use ...	CLIMITS	CBINS
... frequency, use ...	FLIMITS	FBINSIZE, FREF, FBINSCALE
... detector, use ...	DETLIST, LAYERLIST	NOMERGEDETS, NOMERGELAYERS

List of standard keywords:

- AVGTIME – (Input) for RADES and RADPS, the amount of data to average together (expressed in seconds of exposure).
- CBINS – (Input) when procedure allows multiple simultaneous light curves or power spectra, this specifies the energy bands of each ($2 \times N$; units in channels).
- CHANAVG – (Output) for energy spectra, the channel label for each spectral bin (channels).
- CHANWIDTH – (Output) for energy spectra, the channel width for each spectral bin (channels).
- CLIMITS – (Input) requested energy limits (2-vector; units in channels).
- DETLIST – (Input) list of detectors to process (for PCA, a numerical vector with values in the range 0-4).
- ERROR – (Output) uncertainty estimate of main output value (with same units as output value).
- ERRTYPE – (Input) specifies the type of error to assign to energy or power spectra.
- EXPOSURE – (Output) exposure of each output time sample (seconds).
- FLIMITS – (Input) specify the frequency range of interest (2-vector, units in Hz).
- FBINSCALE – (Input) when rebinning in frequency, specifies either 'LIN'ear or 'LOG'arithmic rebinning. Do not use with FBINSNEW.
- FBINSIZE – (Input) when rebinning in frequency, specifies the width of a bin at frequency FREF (Hz). Do not use with FBINSNEW.
- FBINSNEW – (Output) when rebinning in frequency, specifies an explicit list of requested frequency bins ($2 \times N$; units of Hz).
- FBINSUM – (Output) the number of original frequency bins that have been rebinned into each output power spectrum frequency bin.
- FREF – (Input) when rebinning in frequency, specifies the reference frequency at which the bin width FBINSIZE is measured (Hz). Do not use with FBINSNEW.
- FREQAVG – (Output) the frequency label of each power spectrum frequency bin (Hz).
- FREQWIDTH – (Output) the frequency width of each power spectrum frequency bin (Hz).
- GTI – (Input) requested time intervals for data processing ($2 \times N$, MET seconds).

- GROUP_BY – (Input) specifies how good-time intervals should be grouped into spectral intervals.
- LAYERLIST – (Input) list of detector layers to process (for PCA, a string vector specifying layer names such as 'X1L', 'X1R', 'X2L', 'X2R', 'X3L', 'X3R').
- MINFRACEXP – (Input) specifies the minimum fractional exposure for energy and power spectra (units: fraction of the spectrum duration).
- MAXGTIGAP – (Input) maximum allowed gap between good time intervals (seconds).
- MINGTI – (Input) the minimum allowed good time interval (seconds).
- NFBINS – (Input) the number of requested frequency bins. (specify only one of NFBINS, NYQUISTFREQ, SAMPFREQ, TBINSIZE)
- NOCROSSGTI – (Input) if set, then spectra are not allowed to be averaged over a gap between good time intervals, no matter what the setting of AVGTIME.
- NOMERGEDETS – (Input) if set, then individual detector counts are processed separately, and returned separately.
- NOMERGELAYERS – (Input) if set, then individual detector layers are processed separately, and returned separately.
- NSPECSUM – (Output) upon return, gives the number of original power spectra that were averaged to make each output spectrum.
- NYQUISTFREQ – (Input) requested nyquist frequency of power spectrum (Hz). (specify only one of NFBINS, NYQUISTFREQ, SAMPFREQ, TBINSIZE)
- OUTFILENAME – (Input) a string specifying the output file name. If no name is given, then no output file is written.
- OUTFORMAT – (Input) specifies the output file format (procedure-dependent).
- OUTUNITS – (Input) specifies the units of the main output variable.
- OVERLAPMAX – (Input) the maximum time overlap of the input files (seconds).
- P0 – (Output) the mean count rate for each output spectrum (count/second).
- QUALITY – (Output) quality label for each output value (0=good; else=bad).
- QUIET – (Input) if set, then only print warnings or errors.
- SAMPFREQ – (Input) requested light curve sample frequency (Hz). (specify only one of NFBINS, NYQUISTFREQ, SAMPFREQ, TBINSIZE)
- STATUS – (Output) upon return, a value of 1 indicates success, 0 indicates failure.

- STEPTIME – (Input) specifies the time increment between consecutive spectra (seconds).
- TBINSIZE – (Input) requested time bin size for light curve samples (seconds). (specify only one of NFBINS, NYQUISTFREQ, SAMPFREQ, TBINSIZE)
- TIME – (Output) time label for each output sample (MET seconds).
- TLIMITS – (Input) requested time limits of data processing (2-vector, units of MET seconds).
- TPIXR – (Input) specifies how the output time bins should be labeled (0=beginning of bin; 0.5=center of bin; 1.0=end of bin).

10 Appendix: Installation

This section provides instructions on how to download and install the IDL Extract software package.

You must first have the IDL Astronomy library installed on your computer. You can find the complete library at the website
`http://idlastro.gsfc.nasa.gov/`

The IDL Extractor packages is also available from the IDL Astronomy web site, in the "contribution" section. It can be found at the following location:
`http://idlastro.gsfc.nasa.gov/ftp/contrib/rxte/`

You may download either a tar or a zip archive file, whichever you prefer and are comfortable with. Expand the archive with the appropriate software into its own directory.

You must set your IDL path variable (`IDL_PATH`) properly. It should include the directory where the IDL Extract program files can be found, and should also be prepended with a "+" symbol. For example, this may be the appropriate thing to do:

```
csh> source /usr/local/bin/idl_setup
csh> setenv IDL_PATH +/home/craigm/lib/idl_extract:$IDL_PATH
```

where, of course, `/usr/local/bin` and `/home/craigm/lib/idl_extract` are specific to your setup.

IMPORTANT NOTE: make sure that the IDL Extract directory appears *before* the IDL Astronomy library in your `IDL_PATH`. There are several IDL Extractor procedures that supercede standard library routines, in order to be able to read gzipped files natively. If the path is listed in teh wrong order, then you will not be able to open gzipped files, and will have to unzip them by hand before running any IDL Extractor tools.