Yaxx Reference Manual

This document describes the **yaxx** tool. It includes discussion of usage, configuration parameters, and analysis results. For a quick start guide including a tutorial example see the Quick Start guide. For installation instructions see the Install guide. Throughout this document the variable $YAXX is assumed to be the directory in which **yaxx** was installed.

Overview

Yaxx is a Perl script that facilitates batch spectral processing of Chandra data using CIAO tools, *Sherpa*, and Perl open source software. It includes automated spectral extraction, fitting, and report generation. The primary emphasis is on having a simple tool that can be run without requiring an extensive learning curve. The simplest task of fitting a default model to one source in a field requires only three steps: copy the *Chandra* data into an appropriate directory, create a two-line file specifying the ObsId and source position, and run **yaxx**. However, for those with the motivation, **yaxx** is highly configurable and can be customized to support complex analysis. In particular **yaxx** uses template files and takes full advantage of the unique *Sherpa* / S-lang environment to make much of the processing user configurable.

The basic analysis flow using **yaxx** is:

- **Enter the CIAO environment**
- **Create a yaxx analysis root directory**
- **Assemble input *Chandra* data in directories organized by ObsId**
- **Create an object list file defining sources to be processed**
- **Adjust configuration files, e.g.**
  - **Define spectral fit models**
  - **Change data grouping specifications**
  - **Change fit method and statistic**
- **Run yaxx**
- **Examine analysis output data files**
- **Generate summary fit results data table**

This manual is organized by describing each of the components in the analysis flow given above. Much of the detailed functionality of **yaxx** is specified by the configuration parameters. To avoid getting bogged down, throughout the manual we refer to relevant parameters in general terms, while the details are given in the alphabetically organized Configuration Parameters section.

Throughout this document the variable $YAXX is assumed to be the directory in which **yaxx** was installed. For instance, if you install **yaxx** in your home directory as shown in the installation guide, you would do (for csh or tcsh):

```
set YAXX=~/yaxx
```

A quick start guide is available (doc/quick_start) which will guide you through a simple example and demonstrate most of the key features needed to run **yaxx**.

Citation

If you make use of **yaxx** in analysis for published results, it is requested that you include an appropriate citation. Shown below is an example if you are using the the natbib bibliographic package:

```
\bibitem[Aldcroft (2006)]{Aldcroft06} Aldcroft, T.L. 2006, Yaxx: Yet Another
X-ray Xtractor
```

At the point of citation please include a text footnote with the **yaxx** home page URL:

```
http://cxc-www.harvard.edu/contrib/yaxx
```

CIAO environment

Yaxx must be run from within the CIAO environment. Source the appropriate initialization script (as instructed in the CIAO documentation) to enter the CIAO analysis environment.

Note that if the FTOOLS package is used within the same session as CIAO then the FTOOLS initialization must be done *before* CIAO initialization. Search for ‘‘Other Software Packages’’ in the CIAO documentation for further details.

Analysis Root Directory

Yaxx is run from an analysis root directory that should be distinct from the $YAXX source installation directory. The analysis directory contains the project configuration parameters, the object list file, and the output analysis products in subdirectories by ObsId. These are each described in subsequent sections.

Chandra Input Data

The *Chandra* input data for running **yaxx** are:

- **ACIS event file (acis?*_evt2.fits*) [required]**
- **Aspect solution file or files (pcad?*_asol1.fits*) [required]**
- **Source detect file (acis?*_src2.fits*) [optional]**
- **ACIS bad pixel file (acis?*_bpix1.fits*) [optional]**

The location of these files is defined by the `input_dir` configuration parameter. By default **yaxx** looks in the analysis directory in sub-directories named *obs<ObsId>*. In general `input_dir` can specify either an absolute or relative path name, and includes specification of the ObsId formatting (for instance whether the ObsIds are 5 digits with leading zeroes).

The input file names which **yaxx** searches for are given by the four ‘‘glob’’ parameters `evt2_glob`, `asol_glob`, `src2_glob`, and `badpix_glob`. Each of these specify a file match template using the Unix wildcard characters ’*’ and ’?’. The default values listed above match the file naming convention for data downloaded from the *Chandra* archive and allow for gzipped files.

When starting to use **yaxx** one should decide whether to store the input data in a separate repository or to have the input data co-located with the analysis results. The latter is selected by default. This is a matter of preference, but if you foresee defining different samples or different projects based on a common set of data then it makes sense to use a distinct data repository.

Yaxx can handle input data files that are gzipped, but for processing efficiency (to avoid repeatedly unzipping a large event file) it always works with an unzipped version on disk. Thus if the input file is gzipped **yaxx** will generate a local unzipped version. In general the best practice is to unzip the input data files in which case **yaxx** simply creates a link to the file.

One caveat when input data are stored in the same directories as the output results is that **yaxx** will create files named *acis_evt2.fits*, *pcad_asol1.fits*, *acis_src2.fits* and *acis_bpix1.fits*. It is important that the input data files and file template parameters do NOT match those names. If the default settings are used there will be no conflict.

Note that the requirement of supplying aspect solution files is expected to be removed as support for the *specextract* tool is added to **yaxx**.

Object List File

The list of ObsIds and sources, along with other source-specific information such as position and redshift, is specified as tabular data in the object list file. The name of the file is given by the `objlist` parameter. The format of this file can be FITS, RDB, HTML, or ASCII (with several common column delimiters supported). **Yaxx** will automatically detect the table format and read the table. Details on the table format are given in the Object List File Format subsection.

As an example, an ASCII table could consist of the following lines:

```
obsid   src     redshift        X       Y      object
3102    1       0.32          4167   4085     Q1250+568
 877    1       0.22          4200   4102.1   ’Source 82’
```

This is a space-delimited table and indicates that the first source is named Q1250+568 with physical sky coordinates (4167,4085). Any fits that rely on a redshift will use 0.32. The second source is ‘‘Source 82’’. Note the quotes for this space-delimited file. In an RDB (tab-separated) file the quotes would be unnecessary.

The only column in the object list file that is always required is the `obsid`. However, **yaxx** must be told where the sources are by one of three methods:

- **Specify the physical sky coordinates in columns `X` and `Y`**
- **Specify the RA and Dec in columns `ra` and `dec`**
- **Supply a *wavdetect* or *celldetect* FITS file (e.g. acis*src2.fits from the *Chandra* primary products), and do NOT include any of the columns `X`, `Y`, `ra`, or `dec`. In this case you typically supply a `src` number to indicate which source the file to process (see examples below).**

Examples

No source number

The `src` column is not required. In this case the source numbers are automatically incremented from 1 for each obsid:

```
obsid   redshift        X       Y         object
3102    0.32          4167.4  4085.2     Q1250+568-A
3102    0.32          4706.4  3916.7     Q1250+568-B
 877    0.22          4378.5  3892.4     'Source 82'
```

Using detect file with source number

```
obsid  src  redshift  object
3102   2     0.32     Q1250+568-B
 877   3     0.22     'Source 82'
```

The `src` field given here corresponds to the `component` column in CIAO *wavdetect* or *celldetect* files.

Using detect file with no source number

All sources in the detect FITS file which have `net_counts` >= `min_counts` (**yaxx** parameter) will be processed:

```
obsid  redshift  object
3102    0.32     Q1250+568
 877    0.22     'Source 82'
```

Different delimiter

Here an ASCII data file uses the '|' character to mark the columns. In this case the extraction radius (pixels) has been explicitly specified:

```
obsid |  redshift |     X    |   Y     |  object       | rad
3102  |  0.32     |     4167 |  4085   |  Q1250+568-A  |  9
3102  |  0.32     |     4706 |  3916   |  Q1250+568-B  | 14
 877  |  0.22     |     4378 |  3892   |  'Source 82'  | 12.5
```

Minimal object list file

The minimal object list file consists of just the `obsid` and requires that a detect FITS file be available:

```
obsid
 3102
```

This will process every source in the detect files for the specified list of ObsIds. A more typically useful example is to include the **src** field as well to pick out the desired sources.

Yaxx columns

The column names below have special meaning in **yaxx** and are directly used in processing. The names are case-sensitive.

obsid

Chandra Observation ID

src

Yaxx source identifier which corresponds to the number in the source directory name. This must be a positive integer. If not supplied then numbering within each ObsId starts at 1 and increments.

component

Synonym for `src`, which is useful if using a detect file as the object list. A detect file could be produced by running `wavdetect` or `celldetect`, or be a *Chandra* source file (acis*src2.fits) from standard processing.

X, Y

Physical sky coordinates (pixels) of the object. If not supplied, **yaxx** will use (`ra, dec`) or the detect file to determine (`X, Y`).

ra, dec

Celestial sky coordinates (degrees J2000) of the object. If not supplied, **yaxx** will use (`X ,Y`) or the detect file to determine (`ra, dec`).

rad

Circular source extraction radius (pixels). If not supplied and no source region file is present, **yaxx** will determine an extraction radius for each object based on the off-axis angle and the `psf_fraction` and `psf_energy` parameters.

Of the columns listed above, only `obsid` is always required. If a column is supplied, then defined values must be present for all sources (i.e. one cannot leave a numerical column value blank for any sources).

Non-yaxx columns

All column data in the object list file is read in and associated with each object. Within the configuration files those object data can be accessed with the syntax `%VALUE{<col_name>}%`. The most common use for this is to supply fit model parameters specific to each object such as redshift, Galactic column or abundance. For instance, the default absorbed power-law model in **yaxx** includes the *sherpa* commands:

```
source = xsphabs[gal] * xszwabs[abs] * pow[pow1]
gal.nh       = %VALUE{gal_nh}%
abs.redshift = %VALUE{redshift}%
```

This has the effect of setting the Galactic neutral hydrogen column and absorber redshift to the column values of `gal_nh` and `redshift` in the object list file. Note that as a special case, if `gal_nh` is not given then **yaxx** will automatically determine an appropriate value for an extragalactic source at the given object sky position.

Object List File Format

FITS

This must be a FITS compliant table file (ASCII or binary). The first table in the file will be used. The most common use of a FITS object list file would be *celldetect* or *wavdetect* source output file.

ASCII

This must be a plain text file where the first non-comment line specifies the column names. Any line with '#' as the first character is considered to be a comment. For an ASCII table **yaxx** attempts to guess the column delimiter, trying (in order) Comma, Ampersand ‘‘&’’, Vertical bar ‘‘|’’, Tab, and Whitespace. The first one which gives a sensible table is used. A ‘‘sensible table’’ means that there are at least two columns and every row has the same number of columns. Within the table one can use single or double quotes to have an entry that includes the column delimiter.

RDB

This must be a standard RDB format file, which has a column name row, a column format row, and tab separated values. Any line with '#' as the first character is considered to be a comment.

HTML

This must be standard HTML (whatever that might be) with a table that can be parsed by the `HTML::TableParser` routine. The first table in the file will be used.

Configuration Files

Overview

The operation of **yaxx** is controlled by configuration files that define variables and data structures used by **yaxx**. In order to separate system parameters from those typically modified by users (while still maintaining full configurability) **yaxx** reads a heirarchy of configuration files. In order of increasing precedence these files are `System`, `Project`, `ObsId`, `Source`. The `System` file and `Project` files are both required, while the `ObsId` and `Source` files are optional. Each configuration file must be named *yaxx.cfg*, with the file location determining its scope:

System

System-wide configuration data located in the **yaxx** installation directory $YAXX. All **yaxx** runs read this file, which for basic usage should not need modification. This file includes the default report layout, file naming specifications, the basic sherpa fitting script, and a baseline set of fit models.

Project

Configuration data for each yaxx analysis ‘‘project’’. This file is located in the analysis root directory and typically includes basic run parameters, custom fit models and rules, and report formatting rules. Any item in the system-wide yaxx.cfg file can also be overridden here. Normally one would start a project by copying *$YAXX/User/yaxx.cfg* into the analysis directory and then modifying that file as needed.

ObsId

Configuration data specific to a particular obsid, located in *<`output_dir`>/obs<obsid>/yaxx.cfg*. This allows specification of different run parameters for a particular ObsId.

Source

Configuration data specific to a particular source, located in *<`output_dir`>/obs<`obsid`>/src<`srcid`>/yaxx.cfg* >. One reason to use this is to change the fit models for a particular source in the sample, for instance to add an iron line or fit a thermal model.

One could also create a source-specific fit model with particular starting fit parameters that help with convergence. In large batch-fitting applications there are typically some oddballs that benefit from fine tuning.

Configuration file format

These configuration files are specified in a simple format that allows for specification of both simple parameter values as well as complex data structures. Users should have no difficulty modifying the self-documented files by example, but for details refer to the `Config::General` documentation contained in the **yaxx** installation:

```
 perldoc $YAXX/yaxx-perl/Config/General.pm
```

One issue to note is the use of the so-called here-doc notation to specify a multiline string. For instance a model specification could be given by the following, which sets the model parameter `pl` to the multiline string enclosed by the matching END_MODEL tags. The matching tag name END_MODEL is arbitrary as long as it does not appear in the parameter string.

```
  pl <<END_MODEL
    source         = pow[pow1] * xswabs[gal]
    gal.nh         = %VALUE{gal_nh}%
    pow1.gamma.min = -1.5
    pow1.gamma.max = 3.5
  END_MODEL
```

Project parameters

Here we describe in detail each of the parameters that are found in the `Project` level configuration file by default. These are parameters that will typically be adjusted by the user for an analysis project.

input_dir

This controls the location of the input ACIS, PCAD (aspect solution), and source (detect) data files. The string is used as input to a C-style *sprintf* command, with obsid as the parameter, to generate the actual directory. See the *sprintf* man page for details. For instance, with `input_dir=my_data/obs%05d/inputs`, the input data for obsid 956 would be expected in *./my_data/obs00956/inputs/*. With `input_dir=obs%d`, the files would be in *./obs956/*. The input and output dirs can be the same.

output_dir

Output data goes here, in directories named *obs<`ObsId`>*.

objlist

Name of the object list file

log_file

The name of a directory (if ending in a ‘‘/’’) or file for logging the yaxx run results. If it is a directory, the log file is automatically named using the date and time of the run.

loud

Run **yaxx** loudly if set to 1. This should typically be enabled.

projection

Do projection in Sherpa to calculate the parameter confidence intervals. This is more robust (in terms of accuracy) but slower than uncertainty.

uncertainty

Do uncertainty in Sherpa. If results for both projection and uncertainty are available, **yaxx** will use projection.

unbinned_method

Sherpa fit method for unbinned data. See `ahelp method` for options.

unbinned_stat

Sherpa fit statistic for unbinned data. See `ahelp statistic` for options.

binned_method

Sherpa fit method for binned data. See `ahelp method` for options.

binned_stat

Sherpa fit statistic for binned data. See `ahelp statistic` for options.

min_energy

Minimum energy for fitting (eV). The derived quantity `min_energy_kev` can be used via %VALUE{min_energy_kev}% within template files, but it cannot be set directly in the configuration file.

max_energy

Maximum energy for fitting (eV). The derived quantity `max_energy_kev` can be used via %VALUE{max_energy_kev}% within template files, but it cannot be set directly in the configuration file.

min_src_rad

Minimum source extraction radius (pixels) applied to automatic determination of the radius using the Enclosed Counts Fraction tables.

bgd_ann_sep

Separation between source circle and inner circle of background annulus (pixels)

bgd_ann_wid

Background annulus width (pixels)

min_counts

Minimum broad band counts for **yaxx** to process a source. This can be useful if processing sources from a *detect* FITS file.

group_val

Value used for grouping spectra. Grouping is done within *Sherpa* as specified in the appropriate fit template file (*$YAXX/resources/fit_grouped* by default). See `fit_rules` and `ahelp groupByCounts` for additional information. Currently available *Sherpa* S-lang routines for grouping are:

```
groupByCounts( [dset,] numCounts )
groupBySNR( [dset,] minSNR )
groupAdaptively( [dset,] minCounts )
groupAdaptiveSNR( [dset,] minSNR )
```

Use the existing *fit_grouped* template as an example for selecting different grouping options.

spec_chan_type

Extract spectrum as a ’pi’ or ’pha’ channel type.

psf_fraction

If no extraction radius is explicitly supplied, **yaxx** automatically determines the radius so that it includes this specified fraction of the PSF at the `psf_energy`. This makes use of the HRMA circular Enclosed Counts Fraction tables supplied by the CXC Calibration group.

psf_energy

Determine extraction radius for at energy (keV)

fit_rules

The rules which specify different models and/or fit binning based on the number of source counts are given here. This table must have the three columns `model_name`, `Fit file`, and `condition`, in that order.

```
#---------------------------------------------------------------------------
# Rules defining which models to fit
#---------------------------------------------------------------------------
fit_rules  <<END_FIT_RULES_TABLE
  # model_name      Fit file                  condition
  # ---------   -------------------         -------------
  pl              fit_ungrouped             counts <= 100
  pl_fix_abs      fit_ungrouped             counts <= 100
  pl              fit_grouped               counts >  100
  pl_fix_abs      fit_grouped               counts >  100
  pl_abs          fit_grouped               counts >  200
END_FIT_RULES_TABLE
```

- ***model_name***
  Name of the model as given in the model definition parameter

- ***Fit file***
  Name of the *Sherpa* fit template file in the *resources* directory within the **yaxx** installation directory. This specifies the script used by *Sherpa* to do the spectral fitting and to create the output files. Users are encouraged to inspect and customize these templates as needed. It would be wise to give customized files a new name to avoid confusion with the **yaxx** distribution files.

- ***condition***
  Boolean condition which must be satisfied for the fitting rule to be applied. This allows different models and grouping to be applied based on the source attributes. In the given example, a spectrum with 50 counts would be fit unbinned with each of the models `pl` and `pl_fix_abs` (which are a powerlaw and absorbed powerlaw with Gamma fixed, respectively). A spectrum with 1000 counts would be fit by `pl`, `pl_fix_abs`, and `pl_abs` (absorbed powerlaw).

  The condition can be more complex and include tests using any of the columns in the object list file. For instance, one could have a condition

  ```
  (counts > 200 and redshift < 1) or (counts > 500) or obsid == 3102
  ```

model

The models to be fit are given as Sherpa commands as seen in the example below. Each model must have a unique name and be defined within the `<model> ... </model>` structure in the configuration file. Of note is the usage of the `%VALUE{gal_nh}%` variable to specify the Galactic column (in units of 10^20 / cm^2). This value is automatically determined by yaxx using colden unless supplied in the object list file. The `%VALUE{redshift}%` will default to zero if not supplied.

```
#-----------------------------------------------
# Power law with redshifted intrinsic absorption
#-----------------------------------------------
pl_abs <<END_MODEL
  source        = xsphabs[gal] * xszwabs[abs] * pow[pow1]
  gal.nh        = %VALUE{gal_nh}%
  abs.nh        = 0.0
  abs.redshift  = %VALUE{redshift}%
  abs.nh.min    = 1e-4
  abs.nh.max    = 500
  pow1.gamma.min = -1.5
  pow1.gamma.max = 3.5
  freeze gal
END_MODEL
```

common_model_defs

Model component definitions that are always put into script in order to simplify source model definitions. For instance the default for `common_model_defs` includes:

```
xsphabs[gal]
gal.nh       = %VALUE{gal_nh}%
freeze gal
#
pow[pow1]
pow1.gamma   = 1.9
pow1.gamma.min = -1.5
pow1.gamma.max = 3.5
```

This definition of the `gal` model component will be included in every fit script so that a simple powerlaw with Galactic absorption can be defined completely with:

```
    source = gal * pow1
```

System parameters

Here we describe in detail each of the parameters that are in the `System` level configuration file. For basic usage of **yaxx** these parameters will not need to be adjusted. However, those doing more sophisticated analyses or desiring to customize the fit script or output format will need to adjust these parameters. This can be done on either the `System` level file or each `Project` level file. If modifying the `System` file it is recommended that the new specifications simply be added on the end of the file, where they will override the supplied default values. This strategy will make it easier to retain customizations when new versions of **yaxx** are released.

log_time_format
Time format for log file entries. See the `POSIX:strftime` documentation for further details.

timeout
Calls to external tools such as *psextract* or *Sherpa* will time out if they do not complete within the specified number of seconds.

allow_failed_fit
If the *Sherpa* fit for one model fails, continue with others. This is not generally recommended because (if enabled) yaxx will report SUCCESS for source processing even in cases where a one or more fits failed.

fit_plot_size
Size of the spectral fit plot in pixels

use_psextract
Use psextract instead of specextract. Future releases of **yaxx** will allow the choice of using either *specextract* or *psextract* to perform the spectral extraction. At this time only *psextract* is supported and this parameter must be enabled.

src_column_name
Column name in a detect file that identifies the source number. For CIAO *celldetect* or *wavdetect* output files, this is the `component` column.

lock
If enabled, **yaxx** creates a lock file in the source directory for each source while processing. This locks out any other instance of **yaxx** from processing that source. Locking is needed for running multiple instances of **yaxx** (taking advantage of multiple processing) on a single project.

evt2_glob
Name template for the required ACIS event file in the input directory. It can contain Unix file wildcard characters ? and *, and can have multiple values separated by whitespace. This must match exactly one file in the input directory.

asol_glob

Name template for the required PCAD aspect solution `file(s)` in the input directory. This can match multiple files in the input directory, in which case the files will be concatenated in order.

src2_glob

Name template for the optional *wavdetect* or *celldetect* FITS source file in the input directory. This must match either zero or one files.

badpix_glob

Name template for the optional ACIS bad pixel file in the input directory. This must match either zero or one files.

report_index_col

List of columns that are included in the **yaxx** HTML report summary index.

source_image

This structure defines parameters controlling the generation of the 2-d color image of the source events and extraction regions. The parameters and default values are given below.

```
field_of_view = 100       # Image FOV (ACIS pixels)
size_inches   = 3.0       # Plot window physical size (inches)
size_pixels   = 240       # Plot window physical size (pixels)
label = %VALUE{object}%   # Label to include in image
label_x = 0.5             # X position of label (From 0 to 1)
label_y = 0.92            # Y position of label (lower)
color_table = heat        # Color table name
invert = 1                # Invert color table
charsize = 2.0            # Character size
justification = 0.5       # Text justification (0=left,1=right)
```

formatting_rule

Define how **yaxx** will interpret and format variables and fit values for output. This parameter is a sequential set of named structures following the format:

```
<formatting_rule RULE_NAME>
   FORMATTING_ATTRIBUTES
</formatting_rule>
```

Here `RULE_NAME` can include the Unix wildcard characters * and ?. This makes it easy to specify the formatting for a class of fit model parameters. For instance, there are many absorption models that have a parameter `nH`, so one need only make a rule for `*.nH`. If a particular model needs to be different than the generic version, just add a new rule with the full model name, e.g. `zwabs1.nh`. The rule name is not case sensitive.

The example below `*.nh` illustrates all the features of a formatting rule:

```
<formatting_rule *.nh>
  fmt              %.3f             # sprintf() style format specifier
  mult             1                # Multiply by this value before output
  unit_latex       $10^{22}$        # Units in latex
  unit_html        10<sup>22</sup>  # Units in HTML
  report_lower    %VALUE%           # Use fit value if lower limit not found
  report_upper    %INFINITY%        # Use infinity if upper limit not found
  summary_lower   %VALUE%           # Use fit value if lower limit not found
  summary_upper   9999              # Use 9999 if upper limit not found
  default         0.0               # Default value if not supplied
</formatting_rule>
```

summary_header_key_col

Define the set of keywords that are copied from the MDL file header into the summary FITS file. The `write_MDL_header_keys` parameter defines the commands to insert a number of useful FITS header keywords into the fit output MDL file, and `summary_header_key_col` specifies which of those go into the fit summary file created by running `yaxx -summary <model_name>`.

write_MDL_header_keys

Specify the S-lang commands to insert a number of useful FITS header keywords into the output MDL file within *Sherpa* after fitting is complete. This is a key component where **yaxx** takes advantage of the S-lang environment to provide significant flexibility for customization.

For example, imagine you would like to record not only the model flux within the fit energy bounds (which is done by default), but you would also like the ‘‘de-absorbed’’ flux in which selected absorbing model components have been removed. This would be accomplished as follows:

First, in each source model definition (`model`) set a variable defining the de-absorbed model, for instance:

```
source = gal * abs * (pow1 + gauss1d)  # Standard source definition
deabsorbed_source = "pow1 + gauss1d"   # "De-absorbed" source
```

Note that this is defining a S-lang string variable that must conform to the *Sherpa* model syntax. The actual variable name (`deabsorbed_source` in this case) is arbitrary. Each **yaxx** model definition requires its own definition of the deabsorbed source string.

Second, in the `write_MDL_header_keys` definition add a line to calculate the de-absorbed flux:

```
flux_deabs=get_eflux(1, [%VALUE{min_energy_kev}%, %VALUE{max_energy_kev}%], deabsorbed_source)
```

Then add a line to insert a new MDL file header keyword:

```
fits_update_key (fp, "FLUXDABS", flux_deabs.value, "De-absorbed flux (" + flux_deabs.units + ")")
```

Finally, add a `summary_header_key_col` entry so that the value gets put into the summary file:

```
summary_header_key_col FLUXDABS
```

file_definition

This structure defines the actual file name for all file types used within **yaxx**.

Run yaxx

Yaxx is run from the analysis root directory. For each source in each ObsId the following processing steps are performed:

- **Make output directories if needed**
- **Set lock for source**
- **Start a log file**
- **Copy or link input data to analysis output directory**
- **Make source and background extraction region files**
- **Extract spectrum (PI or PHA file) from event data using *psextract***
- **Make image of source event data with PGPLOT**
- **Fit specified models to spectral data using *sherpa***
- **Make report summary pages in HTML and LaTeX/postscript**
- **Release lock for source**

A processing run is successful if **yaxx** completes all necessary steps and and makes the final report for each source in the specified object list file. Underlying this definition is the idea of file dependencies, which is a key concept in the **yaxx** processing. Each step is run only if the output files for that step are non-existent or are older than the input files. This is similar to the way in which a software package is compiled based on source file dependencies.

Reprocessing

The practical importance of the file dependence concept lies in processing moderate to large samples of sources. It is inevitable that all or part of the sample will need to be reprocessed (probably many times) in the course of doing a serious analysis and writing a paper. In the initial processing there are often issues with one or more sources requiring some fine tuning. After the first look one often sees that fit parameters need to be adjusted. As time passes the CALDB may be updated or after the paper is written the referee may have suggestions about the data analysis.

Yaxx was specifically designed to facilitate reprocessing by examining file dependencies. Instead of manually tracking which sources need to be processed, the normal method is to run **yaxx** on the entire sample. Those with changed input files will be processed while those with no changes will be left untouched.

A caveat to this dependence concept is that updates to the configuration parameters are not tracked (but future versions of **yaxx** should include this functionality). As a consequence, accounting for parameter updates requires that a user manually force the appropriate reprocessing. For example, if a fit model were updated then one would tell **yaxx** to force processing of the fit by cleaning all the output products of the fit step before processing:

```
  yaxx -preclean fit
```

If other ealier dependencies were also unmet the appropriate processing steps would be run as well. The available options are listed in the `-preclean` command line option description.

Spectral fitting with *Sherpa*

Key parameters

The spectral fitting in *Sherpa* depends directly on the parameters listed below. Users should read the documentation for each parameter and understand how they affect the final fit results.

- **`projection`**
- **`uncertainty`**
- **`unbinned_method`**
- **`unbinned_stat`**
- **`binned_method`**
- **`binned_stat`**
- **`fit_rules`**
- **`model`**

Background subtraction and fitting

The default *Sherpa* fitting commands depend on whether the spectra are to be fitted grouped or ungrouped.

If ungrouped, the standard method is to define a *Sherpa* background model and fit to the extracted background PI or PHA spectrum. This fitted background model is then frozen for subsequent fitting of the extracted source region (which contains source + background). The background model that is fit is simply a normalization factor times an empirically derived 8th order polynomial with 6 added gaussians. The coefficients of the polynomial and gaussians are frozen and have been set based on fitting (by the **yaxx** author) of ACIS back-illuminated (S3) or front-illuminated background data (I2,I3,S2) background datasets from the year 2000. These background models have been provided for the convenience of users, but we no guarantees are made of their correctness or applicability to any particular analysis. Users are strongly encouraged to investigate the background fitting and provide feedback to the **yaxx** author if improvements are warranted.

If the spectrum is grouped, the standard method is to use the `subtract` command to subtract the extracted background spectrum from the source spectrum before fitting. This works reasonably well for moderate to high counts spectra and aids in the visual interpretation of spectral plots. However, there are good arguments against doing background subtraction, and users should explore the literature and decide what makes sense for their analysis. If background fitting is preferred, use the example of the ungrouped commands to appropriately modify the grouped commands template.

Command line options

The available command line options when running **yaxx** are:

-obsid <ObsId> [<ObsId2> ..]

Process only the specified ObsId or ObsIds. If multiple ObsIds are desired they must be in a quoted list separated by space, e.g.

```
yaxx -obsid "1232 411 522"
```

-src <src> [<src2> ..]

Process only the specified source or sources. If multiple sources are desired they must be in a quoted list separated by space.

-preclean <file_group> [<file_group> ..]

The typically used values for <file_group> are listed below. In addition, it is possible to specify any of the file types given in the `<file_definition>` block of the system-level yaxx.cfg file.

- **all**
  Start from scratch by removing all output files. This does not touch the source and background region files defined for each source since they may have been manually edited.
- **region**
  Remove the *src.reg* and *bkg.reg* region files in the source directory.
- **extract**
  Spectral extraction processing outputs
- **fit**
  Sherpa fitting outputs
- **source_image**
  Color 2-d image of source and extraction regions from PGPLOT
- **report**
  Final LaTeX/postscript and HTML reports
- **log**
  Log files kept in the source directory.
- **resources_dir**
  Directory containing local copies of **yaxx** resource files such as fit templates, report templates, and HTML images.

-summary <model_name>

This is a special mode of **yaxx** that can be run after all processing has successfully completed and you are happy with the results. For a given fit model, it creates a summary FITS file containing all of the fit parameter values (with lower/upper limits), fit statistics, and object list data. The output is a file named *summ_<model_name>.fits*. Because different models have different fit parameters, this must be run separately for each fit model. For instance the command

```
yaxx -summary pl
```

will produce a file named *summ_pl.fits* with a row entry for each source that was fit with the `pl` model.

Pausing yaxx

For very long processing runs **yaxx** can be told to pause its run in order to free up processor resources, e.g. during the day when users are working interactively on the computer. This is done by creating a file named *yaxx_pause* in either the home or analysis directories. This is most easily done with the Unix *touch* command, which creates an empty file if none exists. **Yaxx** checks for this file at the beginning of processing for each new source, and will wait until the file is removed before continuing.

Yaxx Output Data Files

The analysis data files associated with yaxx processing have the structure:

```
<output_dir>/obs<obsid>/src<src>
```

The files unique to an ObsId are stored in the *obs<obsid* >> directory, and those unique to the particular source are in the *src<src* >> (i.e. the ‘‘source’’ directory). Some of the more useful files in the source directory are:

```
acis*.pi       : The various pi (or pha) spectral files for fitting
acis*.rmf      : RMFs
acis*.arf      : ARFs
<fit_model>.in : Sherpa script used to fit <fit_model>, e.g. pl.in
<fit_model>.mdl: Sherpa MDL file for <fit_model>
report.html    : Final fit summary report (HTML)
report.ps      : Final fit summary report (postscript)
report.tex     : Final fit summary report (latex)
log            : Processing log.  log<n> files are old logs
```

Further analysis in *Sherpa*

The <*fit_model*.in >> file is a very useful starting point for doing more detailed or interactive spectral fitting of a particular source. One can exactly recreate the **yaxx** fitting steps in *sherpa* by doing:

```
cd obs3102/src1
sherpa
use <fit_model>.in
```

The *Sherpa* MDL file contains a full record of the fit for a particular model and can be used to easily recover both the source data and final fit values. This allows the user to easily pick up with interactive fitting from where the **yaxx** fit finished with the *Sherpa* command:

```
read mdl <fit_model>.mdl
```

HTML and postscript summary reports

Assuming the processing is successful, examine the results with

```
firefox <output_dir>/report_index.html
```

where firefox can be replaced by the name of your favorite web browser. This shows the **yaxx** report index which links to report pages for the individual sources. The report page shows a summary of the source parameters, an image of the source and the extraction regions used, a table of spectral fit results, and plots of the spectral model fits. If there are multiple sources then the individual reports are linked together by the arrows in the upper left corner of each report.

A latex/postscript report is also created and can be viewed with:

```
gv obs<obsid>/src<src>/report.ps
```

The LaTeX fit parameter table within *report.tex* can conveniently be inserted into a manuscript to create a table on spectral fit results.

Processing results FITS table

After successfully processing the entire sample it is possible to generate a FITS table with the results of processing for each source in the sample. For a given fit model, **yaxx** will create a FITS file containing all of the fit parameter values (with lower/upper limits), fit statistics, and object list data. The output is a file named *summ_<model_name>.fits*. Because different models have different fit parameters, this must be run separately for each fit model. For instance the command

```
yaxx -summary pl
```

will produce a file named *summ_pl.fits* with a row entry for each source that was fit with the `pl` model.

Other Extractors

ACIS Extract

The ACIS Extract package http://www.astro.psu.edu/xray/docs/TARA/ae_users_guide.html is similar to **yaxx** but has significantly more functionality, developer effort, and documentation. For those with access to IDL this is certainly a tool that should investigated. **Yaxx** is a complementary tool, designed with an emphasis on ease of use and based on free open source software.

Xassist

Xassist (http://xassist.pha.jhu.edu/xassist/manual/xassist.html) is another X-ray extractor that can support automated processing and limited spectral fitting of Chandra ACIS data. This NASA funded project is based on Python scripts and compiled C/C++ code.

Copyright and Licence

Copyright (C) 2006 by the Smithsonian Astrophysical Observatory

This code is released under the GNU General Public License. You may find a copy at http://www.fsf.org/copyleft/gpl.html.

Acknowledgments

TLA gratefully acknowledges support for the development of **yaxx** from NASA under NASA grant NAS8-39073 and CXC archival research grant AR2-3009X. This project would not have been possible without the substantial efforts of many perl module developers and maintainers of the CPAN.