Yaxx Reference Manual

Table of Contents

- 1 Overview
- 2 Analysis Root Directory
- 3 Input Data
  - 3.1 *Chandra* Data
  - 3.2 *XMM* Data
- 4 Object List File
  - 4.1 Examples
  - 4.2 *Yaxx* columns
  - 4.3 Non-*yaxx* columns
  - 4.4 Object List File Format
- 5 Configuration Files
  - 5.1 Configuration file hierarchy
  - 5.2 Configuration file format
- 6 Run *yaxx*
  - 6.1 Reprocessing
  - 6.2 Spectral fitting with *Sherpa*
  - 6.3 Command line options
  - 6.4 Pausing *yaxx*
- 7 *Yaxx* Output Data Files
  - 7.1 Further analysis in *Sherpa*
  - 7.2 HTML and postscript summary reports
- 8 Processing results FITS table
- 9 Citation
- 10 Related software
  - 10.1 ACIS Extract
  - 10.2 Xassist
- 11 Copyright and Licence
- 12 Acknowledgments
- 13 Appendix
  - 13.1 Configuration file macro substitutions
  - 13.2 Configuration parameters

1 Overview

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 installation guide. Throughout this document the variable `$YAXX` is assumed to be the directory in which *yaxx* was installed.

Yaxx is a Perl script that facilitates batch spectral processing of X-ray data using Perl open source tools and commonly available astronomical software (CIAO, SAS, HEAsoft). 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. However, for those with the motivation, *yaxx* is

highly configurable and can be customized to support complex analysis. In particular the *yaxx* processing flow is fully configurable, easily allowing automation of data reduction steps. *Yaxx* includes default processing threads and output templates for Chandra and XMM spectral analysis.

The basic analysis flow using *yaxx* is for X-ray spectral analysis:

- Create a *yaxx* analysis root directory
- Assemble input data in directories organized by ObsId
- Create an object list file defining sources to be processed
- Adjust configuration files, for instance
  - Define processing thread (e.g. *Chandra* or *XMM*)
  - Define data input and output directories
  - Define spectral fit models
  - Change data grouping specifications
  - Change fit method and statistic
- Run *yaxx*
- Examine analysis output data files and results summary
- Generate summary fit results data table as a FITS file
- Publish and cite *yaxx*

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
```

2 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.

3 Input Data

The location of input data (event files and other files required for data processing and analysis) is defined by the input_dir configuration parameter. 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).

It is worth noting a distinction between the input_dir and the output_dir parameters. The former is a direct specification of the input data directory for a given ObsId, while the latter is a specification of the root directory within which output data for *all* ObsIds will be stored.

3.1 *Chandra* Data

The *Chandra* input data for running *yaxx* with the default *Chandra* thread 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 typical configuration is that *yaxx* looks in the analysis directory in sub-directories named `obs<ObsId>`.

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*.

3.2 *XMM* Data

As for *Chandra* the input data directory for *XMM* is given by the input_dir parameter. However, due to the nature of *XMM* data reduction within SAS there is little flexibility in the input file naming convention and none of the `_glob` parameters described for Chandra data are meaningful. For the default XMM thread configuration it is assumed that an `ODF` directory exists within the given ObsId directory and contains the ODF files from the XMM archive, . If the ObsId directory is named simply as the 10-digit ObsId value within the analysis directory (recommended) , then the corresponding input_dir and output_dir values would be:

```
 input_dir = %s              # Input data directory
output_dir = ./              # Output data directory
```

4 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.

For the XMM data processing the 4-digit value of the orbit is also required, so an example object list file would be:

```
obsid       orbit   ra        dec
0147511101  0526    163.58804 57.42903
0147511201  0527    163.58804 57.42903
```

The only columns in the object list file that are always required are the `obsid`, and the `orbit` for XMM data. 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).

Although the object list file can contain multidimensional columns (e.g. the R[2] column in a *celldetect* FITS file), those columns are not read into the source data structure.

4.1 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. The name of this column is defined by the src_column_name configuration parameter.

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.

4.2 *Yaxx* columns

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

obsid
: Observation ID. In the case of non-X-ray data this is just a unique identifier for the data instance being processed.

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`). For non-X-ray data one currently has to supply some values of (`X, Y`) even if meaningless, but the next release removes this requirement.

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).

4.3 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.

4.4 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 column names in the first row, column formats in the second row, and tab separated values in subsequent rows. Any line with ’#’ as the first character is considered to be a comment, but comments should not occur before the first two

rows.

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.

5 Configuration Files

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, Analysis Thread, Project, ObsId, and Source. The System, Analysis Thread, and Project files are required, while the ObsId and Source files are optional. Each configuration file must be named `yaxx.cfg`, with the file location determining its scope.

The most commonly modified configuration file is the Project file. Some key parameters defined in this file are:

mission	Mission (can be `Chandra` or `XMM`)
thread	Processing thread (`Chandra`, `XMM`, or user-defined thread)
input_dir	Input data directory
output_dir	Output data directory
objlist	Object list file
fit_rules	Specify which spectral models and fit scripts to use for each source based on criteria such as number of counts
model	Define spectral fit models

5.1 Configuration file hierarchy

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.

In most circumstances the the System level configuration file should *not* be directly modified. Instead

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 any of the System, Thread, or Project level files. 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.

Analysis Thread

The thread configuration file has settings specific to the analysis thread. The standard threads supplied with *yaxx* are *Chandra* and *XMM*, and define parameters such as the file_definition, source_image and process_step. The latter is key as it fully defines the data processing flow for the thread. Users wishing to alter the processing flow should copy the appropriate thread directory and modify the copy:

```
cd $YAXX/resources
cp -rp Chandra custom_chandra_thread     # or XMM
emacs custom_chandra_thread/yaxx.cfg     # Etc...
```

Project

The configuration file for each *yaxx* analysis "project" 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 or Thread yaxx.cfg files can be overridden here. Normally one would start a project by copying the template Project config file `$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.

5.2 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
```

Configuration examples and explanation

- Set a single-valued parameter

```
verbose_stdout   = 3                # Verbosity for terminal output (stdout)
```

Important Note: If a single-valued parameter is specified more than once within the *same file*, then that parameter is interpreted by the configuration file parser as an array-valued parameter. This will likely cause a crash of *yaxx*.

- Set a multiline single-valued parameter

  Use the "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
  ```

- Set an array-valued parameter

  Specify the parameter multiple times. As demonstrated here the "=" sign shown in previous examples is optional.

  ```
  summary_header_key_col RA
  summary_header_key_col DEC
  summary_header_key_col OBJECT
  summary_header_key_col COUNTS
  ```

- Create a complex nested structure

  The example below creates a parameter named "formatting_rule" that has elements named "ampl" and "cvrfract". Each of these has its own parameter values. These complex structures can be modified or extended by example, taking care to maintain consistency with the existing structure.

  ```
  <formatting_rule ampl>
    fmt         %.2f
    unit_latex  $10^{-5}$
    unit_html   10<sup>-5</sup>
    mult        1e5
    default     0.0
  </formatting_rule>

  <formatting_rule cvrfract>
    fmt         %.2f
    default     1.0
  </formatting_rule>
  ```

6 Run *yaxx*

Yaxx is run from the analysis root directory. For each source in the object list file the processing steps defined by the process_step parameter are carried out. For the *Chandra* and *XMM* threads the major steps are:

- Make output directories if needed
- Start a log file
- Copy or link input data to analysis output directory
- Make source and background extraction region files
- Process event data as needed
- 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

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.

6.1 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. 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.

6.2 Spectral fitting with *Sherpa*

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.

fit_rules	Specify which spectral models and fit scripts to use for each source based on criteria such as number of counts
model	Define spectral fit models
projection	Calculate confidence intervals via *Sherpa* projection
uncertainty	Calculate confidence intervals via *Sherpa* uncertainty
unbinned_method	Fit method for unbinned spectra
unbinned_stat	Fit statistic for unbinned spectra
binned_method	Fit method for binned spectra
binned_stat	Fit statistic for binned spectra

Note that the last four parameters (fit methods and statistics) are only used within the standard fit scripts `fit_grouped` and `fit_ungrouped` supplied with *yaxx*. For user-defined fit scripts there is no requirement that these parameters be examined, and indeed there is nothing magic in the distinction between binned and unbinned.

Background subtraction and fitting

The standard `fit_ungrouped` template (which fits data without applying grouping) defines a *Sherpa* background model and fits 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.

The `fit_grouped` template uses 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.

6.3 Command line options

The available command line options when running *yaxx* are as follows.

-config <file>

Read the project-level configuration data from the specified file instead of the default `yaxx.cfg` in the analysis directory.

-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. If multiple values are specified they must be in a quoted list separated by spaces. The available `<file_group>` are:

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.

6.4 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.

7 *Yaxx* Output Data Files

The analysis data files associated with yaxx processing have (by default) 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). For the *Chandra* thread 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
```

7.1 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
```

7.2 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.

8 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.

9 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: a batch data processing tool for scientists.
```

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

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

10 Related software

10.1 ACIS Extract

The ACIS Extract package http://www.astro.psu.edu/xray/docs/TARA/ae_users_guide.html is similar to *yaxx* but is specifically focused on *Chandra*/ACIS data. Within that scope ACIS Extract has well-developed functionality and documentation. For those working on *Chandra* data 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, user extensibility, and based on free open source software.

10.2 Xassist

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

11 Copyright and Licence

12 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.

13 Appendix

13.1 Configuration file macro substitutions

Yaxx includes a macro substitution capability that is used within the configuration files. This allows the configuration values to reference *yaxx* file names (e.g. the X-ray event file for a source) or their contents, source characteristics (e.g. the redshift or galactic column), a label for a source characteristic (e.g. "Galactic Column" for the `gal_nh` parameter), or a specially defined value that is set within the *yaxx* perl code.

The general format for macro substitutions is:

```
%MACRO_TYPE%
%MACRO_TYPE{var, MACRO_OPT=val,.., source_var=val,...}%
```

The macro must be written on a single line. The meaning and usage of the macro arguments are given below.

%FILE{<file_type>[.ext], MACRO_OPT=val,..., source_var=val,...}%

This macro returns the file name for the given <*file_type*> using the rules in the file_definition parameter. An optional extension may be added to the <*file_type*>. For example `%FILE{report.html}%` returns `%FILE{report}%` with the `.html` extension added. Setting a source variable *source_var=val* allows a source characteristic to be temporarily set during the evaluation of the macro. The most common use of this is in the *XMM* thread where the detector (PN, MOS1, or MOS2) must be given for many of the detector-specific files. The available FILE_OPT macro options are:

File Macro Options

Option	Function
ABSOLUTE=1	Return an absolute file name instead of the default relative file name.
CONTENT=1	Return the contents of the file instead of the name.
PREFIX= <*value*>	Put <*value*> before the file name. Note that the directory path to the file is left unchanged.
SUFFIX= <*value*>	Put <*value*> after the file name.

Examples:

```
%FILE{resources_dir}%             # Directory containing yaxx resources
%FILE{source_image.jpg}%          # JPEG version of source counts image
%FILE{pi,detector=mos1}%          # MOS-1 PI spectrum file
%FILE{pi,PREFIX=.create_pi_}%     # PI spectrum file prefixed with '.create_pi_'
%FILE{src_reg, CONTENT=1}%        # Contents of source extraction region file
```

%VALUE{var_name, MACRO_OPT=val,..., source_var=val,...}%
%FORMAT{var_name, MACRO_OPT=val,..., source_var=val,...}%

This macro returns the value of a source characteristic, which could be either input data explicitly specified in the object list file (e.g. redshift, ra) or values that correspond to *yaxx* methods (e.g. counts, exposure). The latter category includes all configuration parameters, but in general use of these in the VALUE macro is for experts only. The FORMAT macro is just the VALUE macro with default formatting selected with DEFAULT_FMT=1.

File Macro Options

Option	Function
FORMAT=<format>	Return the value as formatted with the *sprintf* style formatting specification. See the *sprintf* man page for details.
DEFAULT_FMT=1	Use the default output format as defined by the formatting_rule for the variable. If none is defined then no special formatting is applied. This option overrides the FORMAT option.

Examples:

```
%VALUE{exposure, detector=pn}%   # Exposure for the PN detector
%VALUE{redshift}%                # Redshift from object list file
%VALUE{thread}%                  # Thread parameter from project config file
%VALUE{gal_nh}%                  # Galactic NH column (computed if not supplied)
```

%LABEL{var_name}%

This macro returns the label defined by the formatting_rule for the variable. No options are recognized by this macro.

Examples:

```
%LABEL{srcid}%                   # Returns 'Source'
%LABEL{ccdid}%                   # Returns 'CCD'
```

13.2 Configuration parameters

Here we describe in detail each of the configuration parameters that control the behavior of *yaxx*.

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. *[Project]*

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. *[Thread (Chandra)]*

badpix_glob

Name template for the optional ACIS bad pixel file in the input directory. This must match either zero or one files. *[Thread (Chandra)]*

bgd_ann_sep

Separation between source circle and inner circle of background annulus (pixels). *[Project]*

bgd_ann_wid

Background annulus width (pixels). *[Project]*

binned_method

Fit method for binned data. *[Project]*

binned_stat

Fit statistic for binned data. *[Project]*

clean_spec

File group definitions for cleaning files before processing. *[System, Thread]*

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
```

[Project]

environment

Define setup required for an analysis environment including *setenv* commands and running a shell script. The analysis environments defined with this parameter are then called out for use via the use_environment parameter which is typically defined at the thread level. An example `environment` setting is as follows:

```
<environment HEAsoft>
  setenv  HEADAS /soft/lheasoft/headas/i686-pc-linux
  shell   csh
  script  source $HEADAS/headas-init.csh
</environment>
```

The `setenv` parameter sets the given Unix shell environment variable to the supplied value. Multiple `setenv` parameters may be supplied. To source an initialization script, define the `shell` (typically sh, bash, or csh) required to run the script and then the actual initialization script command via the `script` parameter. Only one script may be run per environment setting. *[System]*

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. *[Thread (Chandra)]*

file_definition

This structure defines the actual file name for all file types used within *yaxx*. *[System, Thread]*

fit_plot

Formatting specifications for the spectral fit plots:

```
<fit_plot>
  width_inches  = 2.25        # Width in inches (for latex)
  width_pixels  = 250         # Width in pixels (for HTML)
  n_across      = 3           # Number across on page
</fit_plot>
```

[Thread]

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 spectral model to be used in fitting. Thhis corresponds to the names given in the model definition parameter.

Fit file

Name of the *Sherpa* fit template file in the `resources/<thread>` 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

[Project]

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
  label           Absorbing Column  # Label returned by %LABEL{}% macro
  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>
```

[System]

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.
[Project]

html_report_index

Definition of columns in the HTML report index. This consists of an ordered series of *column* definitions that specify the *header* and *value* for that column. *[Thread]*

input_dir

This controls the location of the input data files. For the *Chandra* analysis thread this points to the directory containing the ACIS, PCAD (aspect solution), and source (detect) data files. With `input_dir=data/obs%VALUE{obsid}%` the input data for obsid 956 would be expected in `./data/obs956/`. The input and output dirs can be the same. *[Project]*

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. *[Project]*

log_time_format

Time format for log file entries. See the *POSIX:strftime* perl documentation or the *strftime* man page for further details. *[System]*

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. *[Project]*

min_counts

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

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. *[Project]*

min_src_rad

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

mission

Specify the mission for the data. Currently this can be either ’Chandra’ or ’XMM’. The mission parameter enables *yaxx* to load mission-specific perl code, and is distinct from the thread parameter which only changes configuration files. *[Project]*

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
```

[Project]

objlist

Name of the object list file *[Project]*

output_dir

Output data goes here, in directories named `obs<''ObsId>''`. *[Project]*

process_step

Define a step in the overall data processing flow for each source. The two most common usages are to execute one or more shell commands or else to call an internal *yaxx* method (aka subroutine) to perform some processing (typically the more complex tasks like doing the spectral fitting). The sequential list of these process steps is the fundamental component of a processing thread (e.g. *Chandra* or *XMM*).

One of the key concepts of the processing is the file dependencies. In order to avoid unnecessarily repeating processing steps, many *yaxx* steps include specifications for the target and depend files. The target files are those files that are created via the process step (i.e. the output), and the depend files are those files that are used in the creation (i.e. the input). If any of the target files do not exist the the process step will be run. If all target files exist but some are older than the depend files, then the step will be run. If any of the depend files are missing then the process step automatically fails without running the method or commands.

This is a complex parameter with a number of options that are described below. Some options are specific to a particular method call, for instance the `mkdir` option is meaningful only for a call to the *yaxx* method `make_dirs()`.

name
The name of the process step. This is required for every process step and *must* be unique.

method
The name of a *yaxx* method (subroutine) to call. The other process_step options are passed to the method. There can be only one `method` option in a process_step.

command
A shell command to be executed. The exit value of the command will be tested to determine successful execution of the step. Following the usual Unix convention a return value of 0 is considered successful. Multiple `command` options can be specified within a single process_step, and they will be executed in the order given.

set_var *var*
Used in conjuction with `command` to create and/or set a variable *var* associated with the source. The variable is set to the output of the command(s). Note that the special yaxx columns such as obsid, RA, Dec, X, Y in the object list cannot be set in this way.

dir
Run the process step within the specified directory.

always_run
Always run the process step, even if a prior processing step has failed. Normally once any step fails the processing is cut short. However, certain key steps that have `always_run=1` will run in any case. This allows for proper clean-up and generation of a report even for failed processing.

depend_file
Specify an input file on which processing depends. If any depend file is newer than any target file then the process step will be done. More than one `depend_file` can be supplied.

depend_glob
Similar to `depend_file` but the expression is interpreted as a Unix file expression with wildcards (e.g. acis_*evt2.fits*). More than one `depend_glob` can be supplied.

target_file
Specify an output file of the processing step. More than one `target_file` can be supplied.

delete_file
Delete the specified file just prior to the actual method call or shell commands. This is *not* done if the processing step is not to be done in the case that the target dependencies are already met or other criteria like `run_if_defined` are not satisfied. More than one `delete_file` can be supplied and the file name can contain Unix wildcard characters.

run_if_defined
Run the process step if the named variable is defined. More than one `run_if_defined` can be supplied.

run_if_true
Run the process step if the value is true (non-zero). More than one `run_if_true` can be supplied.

run_if_file
Run the process step if the named file exists and is readable. More than one `run_if_file` can be supplied and the file name can contain Unix wildcard characters.

[Thread]

projection

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

psf_energy

For the Chandra thread, determine extraction radius for at energy (keV). *[Project]*

psf_fraction

For the Chandra thread, 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. *[Project]*

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)
binning       = 1         # Image binning
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)
event_filter = [energy=%VALUE{min_energy}%:%VALUE{max_energy}%]
                          # CIAO filter on evt file for image
```

[Thread]

spec_chan_type

Extract spectrum as a ’pi’ or ’pha’ channel type. *[Project]*

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. *[Thread (Chandra)]*

src_column_name

Column in an auxilliary source FITS file that identifies the source number. For CIAO *celldetect* or *wavdetect* output files, this is the `component` column. *[Thread]*

src_net_cts_name

Column in an auxilliary source FITS file that gives the net counts. For CIAO *celldetect* or *wavdetect* output files, this is the `net_counts` column. *[Thread]*

summary_header_key_col

Define the set of keywords that are copied from the MDL file header into the summary FITS file created for a fit model. 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>`. *[System]*

thread

Sets the analysis thread as discussed in the Configuration file hierarchy section. *[Project]*

timeout

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

unbinned_method

Fit method for unbinned data. *[Project]*

unbinned_stat

Fit statistic for unbinned data. *[Project]*

uncertainty

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

use_environment

Use the specified enviroment as defined by the corresponding environment parameter. *[Thread]*

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 is ignored. *[Thread (Chandra)]*

verbose_master_log

Verbosity level for output to master log file. The levels are:

```
0:  Silent
1:  Program and obsid level info
2:  Process step summaries for each obsid
3:  Top level outputs for each process step
4:  Detailed outputs for each process step
5:  Debug outputs
```

[Project]

verbose_source_log

Verbosity level for output to individual source log file, as defined in verbose_master_log. *[Project]*

verbose_stdout

Verbosity level for output to stdout (the screen), as defined in verbose_master_log. *[Project]*

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
```

[System]