MIT Kavli Institute **Chandra X-Ray Center**

$\mathcal{TG}$*Cat* Software Manual[*]

David Huenemoerder

September 21, 2010

Purpose & Scope

The primary intent for developing a Chandra Grating-Data Archive and Catalog ("$\mathcal{TG}$*Cat*"[1]) is to make grating spectra more visible and accessible to users. The simple web interface provides access to analysis-quality spectral products (binned spectra and corresponding response files), with the addition of summary graphical products and model-independent flux properties tables. Such products and a browser interface make it easy for a user to find observations of a particular object, type of object, or type of observation, to quickly assess the quality and potential usefulness of the spectra, and to download the data and responses as a package if desired.

$\mathcal{TG}$*Cat* represents the collective effort of MIT and CfA CXC scientists and staff interested in definition and production of a catalog and archive of data taken with the Chandra transmission grating instruments (HETG and LETG), who also have significant experience in processing and analysis of grating data. The archive is expected to provide a significant legacy product of high-resolution X-ray spectra.

The reprocessing scripts, which utilize CXC and other publically available software to create the archive, are also available to users. This software can facilitate standard or customized reprocessing from Level 1 archival data to spectra and responses with minimal interaction (analagous to the `psextract`[2] script for automated imaging spectral extraction).

This document describes the software used to create the catalog's primary data archive, and which can also serve to ease and automate end-user processing of individual observations. The following sections give an overall guide to processing, examples, and a detailed reference manual.

Part I of this document presents an overview of the software requirements and use.

Part II of this document is a function reference manual, organized from high- to low-level functionality, and grouped by instrument (or similar function). There are many more functions in the low-level interfaces than would typically be needed by a casual user.

Part III gives longer examples of customized processing, which are intended to provide enough different cases to help a user in building their own applications or solving problems.

For updated version of this document, source code download, and other information, see

`http://space.mit.edu/cxc/analysis/tgcat/index.html`

I thank Mike Nowak, Dan Dewey, Norbert Schulz, and Arik Mitschang for testing (and using) the software.

[*]Latest `tgcat.sl` version: 1.8

[1]`http://tgcat.mit.edu`

[2]`http://cxc.harvard.edu/ciao/ahelp/psextract.html`

Part I
TGCat Processing Guide

1 Introduction

This Part gives an overview of data processing using *TGCat*, describing the files required, example scripts, and descriptions of how they work. The *TGCat* processing programs are a collection of shell and ISIS/*S-Lang* scripts for the following purposes:

1. Download observations from the Chandra archive and configure the data directories to make them ready for processing;
2. Process from Level 1 to Level 2 and responses using CIAO tools;
3. Produce standard summary products for quick review of the data.

Step 1 uses `slsh` and shell scripts, invoked at the UNIX prompt and creates the directory and filename structure used by the processing scripts in steps 2 and 3. Step 2 is done by a suite of ISIS/*S-Lang* scripts which construct and invoke CIAO tools, using parameters determined from the instrumental configuration, or as specified by the user. Since the functions rely on system calls to run CIAO tools, step 2 must be started in a CIAO-configured shell. Step 3 has no dependence on CIAO, and so summary products can be generated from any shell configured for ISIS and S-Lang. Steps 2 and 3 require execution from an ISIS prompt.

The programs have a fairly general interface which supports customization to handle a wide variety of cases, but it is not intended to be a full substitute for the CIAO tool command-line and parameter interface.

1.1 The Simplest Case

Assuming all software components are installed, this is the minimal set of steps to retrieve and process grating data. Here we use "`> `" for the UNIX prompt, and "`isis> `" for the ISIS prompt.

```
> download_obsid  5
> isis
isis> require( "tgcat" ) ;
isis> run_pipe( "path_to/obs_5" );
isis> exit ;
> display path_to/obs_5/summary*.ps &
```

2 Data Preparation: Download; Directory Configuration

`download_obsid id[ id[ id[...]]]`

This is a shell script which downloads one or more observations from the CXC `cdaftp` site (`ftp://cdaftp.harvard.edu/pub/`). This only retrieves the minimal number of files required for reprocessing. After a successful retrieval, the files are uncompressed and sybolic links are made to generic file names.

The script calls an `slsh` script, `chandra-get`, which retrieves a single observation. `Chandra-get` has configurable options to retrieve other types of files, if desired.

Requires:

> *S-Lang*
> `slsh` (part of slang distribution)
> `chandra-get` (included w/ *𝒯𝒢Cat*)
> `pcre` module (part of *S-Lang* distribution)
> `curl` module (`http://www.s-lang.org/modules/curl.html`)

Example:

```
> download_obsid 5 1103 6441
```

Will result in 3 directories being created in your current directory, each containing subdirectories "primary" and "secondary":

```
obs_1103/
obs_5/
obs_6443/
```

Note that `download_obsid` (and its dependencies on the `curl` and `pcre` modules is *optional.* There are other equivalent methods for downloading data, such as *Chaser*, *webChaser*, `wget`, or manual `ftp` to the archive.

The directory configuration is done with the shell-script, `setup_obsdir`, which may be run independently on already-retrieved *Chandra* archival data.

`setup_obsdir dir[ dir ...]`

This is a shell script. Given one or more directories as retrieved from the `cdaftp` site (such as from *Chaser*), decompress the files and create symbolic links to the generic names used by the reprocessing scripts. (Note: this script is automatically called by `download_obsid` after a successful retrieval.)

This script requires the files to be in the directory structure provided by the CXC archive, with subdirectories called `primary` and `secondary`, and with files with the original CXC names.

Example:

```
> setup_obsdir  obs_5 obs_6441 obs_1103
```

Restrictions: multi-OBI directories are not supported. (There are a few of these, but they were discontinued and replaced with multiple observation IDs); Read/write access is required on the observation directory since files are uncompressed and symbolic links formed.

3 Running ISIS/CIAO Reprocessing Scripts

3.1 Prerequisites

The following software items need to be installed on your system — they are not part of the *TGCat* package:

ISIS installation (Version 1.4.9-20 or later; see `http://space.mit.edu/cxc/isis/`)
(Note: the `xspec` module is not required by *TGCat*);
CIAO tools installation (see `http://cxc.harvard.edu/ciao/download/`)

3.2 *TGCat* Files

The primary *TGCat* reprocessing functions are contained in the file `tgcat.sl`, a collection of ISIS/*S-Lang* functions, and in a few independently maintained files:

```
aglc.sl
findzo.sl
fancy_plots.sl
fancy_group.sl
```

These, and the download/configure scripts described in § 2, are included in the *TGCat* distribution. The package can be obtained from:
`http://space.mit.edu/cxc/analysis/tgcat/index.html`

3.3 Quick Start

- Download an observation (as above via `download_obsid`, or via `chandra-get`, or with *Chaser* or *webChaser*);
- If necessary, configure the observation directory (as above, with `setup_obsdir`; already done if `download_obsid` was used);
- Set up a CIAO environment so that tools like `acis_process_events` are in your path. For details see "Starting CIAO" or "Installing CIAO 4.0" at

  `http://cxc.harvard.edu/ciao/threads/intro.html`.
- Set up the ISIS environment (so that the ISIS binary is in your path, the module paths are defined properly, and so that the *TGCat* *S-Lang* files are in the ISIS path). Detailed download/install instructions can be found at `http://space.mit.edu/cxc/ISIS/`. See "Chapter 2" of the manual at `http://space.mit.edu/cxc/ISIS/manual.html`.
- Start ISIS:

  ```
  > isis
   isis>
  ```

- Load the reprocessing scripts:

  ```
  isis> require( "tgcat" ) ;
  ```

  If this fails because paths are not set, let's assume that you put all the 𝒯𝒢*Cat* *S-Lang* scripts files in `$HOME/isis/scripts`. Then you can do this:

  ```
  isis> tgc_path = getenv( "HOME" ) + "/isis/scripts" ;
  isis> prepend_to_isis_load_path( tgc_path );
  isis> require( "tgcat" ) ;
  ```

- Decide whether you need `findzo` or `tgdetect` to centroid the zero order (if ACIS). If the zeroth order is cratered or blocked, and you have a strong frame-shift "streak" and a strong MEG or LEG spectrum, you can use `findzo`:

  ```
  isis> !ds9 path_to_data/obs_nnnn/evt0 &
  ```

- Use the top-level script to reprocess an obsid:

  ```
  isis> run_pipe( "path_to_data/obs_nnnn" ) ;  % tgdetect method
  ```

  or

  ```
  isis> s = set_source_detection_info( "findzo" );
  isis> run_pipe( "path_to_data/obs_nnnn"; detect_info = s ) ;  % findzo method
  ```

  By default, these will center the search at the target position as determined from the event file's header. For more general specification of the detection method and coordinate you can set information and pass it to the top-level script:

  ```
  isis> s = set_source_detection_info( "findzo", "radec", 181.2345, -5.9876) ;
  ```

  The arguments are the method (`"tgdetect"`, `"findzo"`, or `"none"`), the coordinate type (`"pixel"` or `"radec"`), and the coordinates. (See Section 9 for detailed usage.) Then this is applied as in:

  ```
  isis> run_pipe( "path_to_data/obs_nnnn"; detect_info = s ) ;
  ```

  This function will determine the instrumental configuration from a file header, then call the appropriate tools. It will begin with event processing (`acis_process_events` or `hrc_process_events`), filter events, detect the zeroth order, create grating region and mask files, run `tg_resolve_events`, `tgextract`, make responses (grating ARFs and RMFs), make a light curve, and create summary plots and tables. Source detection will be done with the specified method at the indicated location (or header coordinate). The method "none" means no detection algorithm is run and the coordinates are used as the source location.

  You should see each CIAO tool call with parameters echoed to the terminal (sometimes, several to be executed in succession will be displayed first). All input/output will be done in your data directory, `path_to_data/obs_nnnn`.

  Other qualifiers are supported by `run_cfg` to set some configuration parameters. Specifically, `mask_info` for `tg_create_mask` and `extract_info` for `tgextract`. For example, if you want to extract a serendipitous, you can customize the position, sky regions, and extraction widths as follows:

```
isis> s1 = set_source_detection_info("tgdetect", "radec", 181.2345, -53.9876 );
isis> s2 = set_tg_create_mask_info( 40, 40, 40);
isis> s3 = set_tgextract_info(-25, 25; arcsec);
isis> s4 = set_tgre_osort_info(0.2, 0.5);
isis> run_pipe( "path_to_data/obs_nnnn";
     detect_info = s1, mask_info = s2, extract_info = s3, osort_info = s4 );
```

See Section 9 for further details.

- Examine summary products:

```
isis> !display path_to_data/obs_nnnn/*.ps &
```

(assuming you have the *ImageMagick* suite installed; otherwise, use `gv` or the equivalent)

4 Methodology

The function, `run_cfg` is a high-level wrapper which determines the instrumental configuration and runs the appropriate pipeline tools, starting with the "Level 1" events, and ending with spectra and responses. It also generates summary graphics. Lower level functions are also available for finer control and for customized processing to handle special cases. The *TGCat* functions are not intended to be a complete replacement for the UNIX command-line CIAO tool and parameter interface. The *TGCat* *S-Lang* functions make many assumptions about default parameters and file names, but some flexibility has been provided. The high-level wrappers will take a directory name, move to that directory, and run the low-level tools. The low-level tools work in the current directory.

Example: Perform "Level 1" processing - `acis_process_events`, followed by filtering:

```
isis> run_acis_1( "Data/obs_111" ) ;
```

Example: Perform "Level 1.5" processing - from grating source detection, through responses, starting with the Level 1 event file:

```
isis> run_tg_hrc_1a( "Data/obs_6441" ) ;
```

Low-level functions require execution with the current working directory being the data directory. They typically run one CIAO tool, using default parameters.

Example: Run `acis_process_events` in the current directory:

```
acis_process_events;
```

Example: Run `tg_resolve_events` in the current directory:

```
tg_resolve_events;
```

Some functions take arguments for parameters which very probably need to be specified, since they depend upon details of the observation.

Example: Create a custom grating mask for a source at sky pixel location, $(x, y) = (4096.5, 4100.2)$, with zeroth order radius of 30 pixels, HEG region width of 10 pixels, and MEG region width 20 pixels. Then run `tg_resolve_events` using that mask (all doing i/o in the current directory):

```
isis> src_x  = 4096.5 ;
isis> src_y  = 4100.2 ;
isis> zo_radius = 30 ;
isis> heg_width = 10 ;
isis> meg_width = 20 ;
isis> tg_create_mask( src_x, src_y, zo_radius, heg_width, meg_width );
isis> tg_resolve_events;
```

Example: Compute responses (grating ARF s and RMF s) for orders -1 and 1, using information in the assumed existing event and spectrum files:

```
isis> orders = [-1,1] ;
isis> make_responses( orders );
```

Most functions echo the CIAO command-line before they execute it to show the relevant parameters.

Example: Run `tg_resolve_events`. Note that every line without an ISIS prompt is verbose output:

```
isis> tg_resolve_events;

punlearn tg_resolve_events

tg_resolve_events \
  mode='hl' \
  verbose='0' \
  outfile='evt2' \
  acaofffile='@asol.list' \
  eventdef=')stdlev1_HRC' \
  regionfile='reg1a' \
  infile='evt1' \
  osipfile='none' \
  clobber='yes'
isis>
```

Customization can be done with an optional argument supported by most low-level functions.

Example: Define custom parameters for `tg_resolve_events`:

```
isis> np = make_newpar( ; regionfile="reg1a_test", verbose=5 );
isis> tg_resolve_events( np ) ;
```

```
punlearn tg_resolve_events

tg_resolve_events \
  mode='hl' \
  verbose='5' \
  outfile='evt2' \
  acaofffile='@asol.list' \
  eventdef=')stdlev1_HRC' \
  regionfile='reg1a_test' \
  infile='evt1' \
  osipfile='none' \
  clobber='yes'
```

Note the leading ";" in the `make_newpar()` function – it is *required*: it is how *S-Lang* distinguishes positional arguments (preceding the ";") from the following qualifiers. The `np` container can also be constructed via

```
isis> np = Assoc_Type[ Any_Type ] ;
isis> np[ "regionfile" ] = "reg1a_test" ;
isis> np[ "verbose" ] = 5 ;
```

Any number of replacement parameters can be given, and they will overlay the defaults. No parameter checking is done, however, before the command is invoked; that is deferred to CIAO's parameter interface.

4.1 A Warning

The `infile` *and* `outfile` *parameters, should generally NOT be changed!*

The *TGCat* functions sometimes construct input file names by appending *data model* virtual file specifications ("filters"), and also expect input files of particular names. If custom filtering or user-define file names are required, direct use of CIAO tools may be a better approach.

With experience, names can be changed, but it depends on the tool whether this is simple or requires care. Look at the echoed CIAO parameters in each case to be certain of the usage.

5 Summary Products

Quick-look plots and summary tables can be made with a few functions, or all with one wrapper. The summary functions do not require a CIAO environment - they rely only on ISIS scripts.

Example: Generate summary products with the prefix, `summary` in the same directory as the data:

```
tg_summary( "Data/obs_111", "Data/obs_111/summary" ) ;
```

6 Cleanup

There are some functions to remove unnecessary files and to move the products to another directory. This can be useful to preserve the input data for successive runs with different parameters (e.g., to extract a second source from the same observation).

Example: Remove temporary products, move processing products to a specified directory:

```
tg_run_cleanup( "/path_to_indir", "/path_to_outdir" ) ;
```

Example: If summary files had a custom prefix of `my_plots` instead of the default `summary`, specify that name:

```
tg_run_cleanup( "/path_to_indir", "/path_to_outdir", "my_plots" ) ;
```

Example: To cleanup temporary products, but keep output products in the same place, just give the same path as input and output:

```
tg_run_cleanup( "/path_to_indir", "/path_to_indir" ) ;
```

7 Additional Information

- ISIS: `http://space.mit.edu/cxc/isis/`
- *S-Lang*: `http://www.s-lang.org/`
- *S-Lang* modules:

  `http://space.mit.edu/cxc/software/slang/modules/`

  `http://www.s-lang.org/modules/`
- CIAO: `http://cxc.harvard.edu/ciao/`

Part II

TGCat Function Reference Manual

8 Overview

There is one primary script file, `tgcat.sl`, which provides

- High level wrappers which set up and run a series of low-level functions. The high-level functions generally allow specification of a data directory from which to read the data, and where products are written. These wrappers detailed operation can be changed with a few state-setting functions to support some non-standard extractions.
- A Library of *S-Lang* functions which setup and call individual CIAO tools. Most functions are rigid in filename assumptions, and work in the current directory, assumed to be the data directory.
- Functions to generate pre-defined summary plots and tables from the standard products. These are for a quick-look overview; there is no flexibility provided for plot customization.

The scripts make assumptions regarding the names of input and output files. Specifically, the input files are named:

`evt0`: the `evt1` file from the archive (here considered to be the "Level 0" input; a new "Level 1" file will be produced);

`bpix1`: bad-pixel file from archive;

`asol _?`: Aspect solution files from the archive (`pcad*asol*`); usually only one, but sometimes a few. The `?` will be substituted with a serial index, e.g. `asol_1`;

`asol.list`: list of `asol_?` files, 1 per line (for "`@`"-file use);

`flt1`: `flt1` file from archive, containing Good-Time-Interval (GTI) tables;

`msk1`: `msk1` file from archive, containing the detector mask regions;

`stat1`: `stat1` file from the archive, containing exposure statistics;

`pbk0`: `pbk0`, the parameter block file from archive, containing detector mode information.

The download script `download_obsid` automatically creates symbolic links for these names. For independently downloaded archive directories, the script, `setup_obsdir`, can be used to create these links. The minimal input directory may look like:

```
> ls -ln
total 64
-rw-r--r--    1 501  501     7 Sep 11  2007 asol.list
lrwxrwxrwx    1 501  501    37 Sep 11  2007 asol_1 -> primary/pcadf080302672N003_asol1.fits
lrwxrwxrwx    1 501  501    37 Sep 11  2007 bpix1 -> primary/acisf00005_001N003_bpix1.fits
lrwxrwxrwx    1 501  501    38 Sep 11  2007 evt0 -> secondary/acisf00005_001N003_evt1.fits
lrwxrwxrwx    1 501  501    38 Sep 11  2007 flt1 -> secondary/acisf00005_001N003_flt1.fits
lrwxrwxrwx    1 501  501    38 Sep 11  2007 msk1 -> secondary/acisf00005_001N003_msk1.fits
lrwxrwxrwx    1 501  501    38 Sep 11  2007 pbk0 -> secondary/acisf080302613N003_pbk0.fits
drwxr-xr-x    4 501  501   136 Sep 11  2007 primary
drwxr-xr-x    8 501  501   272 Sep 11  2007 secondary
lrwxrwxrwx    1 501  501    39 Sep 11  2007 stat1 -> secondary/acisf00005_001N003_stat1.fits
```

The files output by the *TGCat* reprocessing scripts are:

`obs_config.txt`:	basic configuration info, from a header;
`evt0.par`:	temporary observation file, output of dmmakepar;
`evt1`:	filtered output of `acis_process_events` and `dmcopy` filters;
`evt1 _0`:	temporary `evt1` – output of `acis_process_events`;
`evt1 _1`:	temporary `evt1` – output of `dmcopy`;
`evt1 _2`:	temporary `evt1` – output of `dmcopy`;
`src1a`:	output of `tgdetect`;
`reg1a`:	output of `tg_create_mask`;
`evt2`:	output of `tg_resolve_events`;
`pha2`:	output of `tgextract`;
`pha2 _bg_-1`:	background spectrum for LETG/HRC, order -1;
`pha2 _bg_1`:	background spectrum for LETG/HRC, order +1;
`pha2 _bg`:	ba ckground spectrum (if LETG/HRC); both orders;
`lc`:	light curve, output of `aglc` (if ACIS) or `dmextract` (if LETG/HRC);
`lc_bg`:	light curve, background (if LETG/HRC);
`findzo .ps`:	(optional) output of `findzo` (if ACIS);
sk`.asphist`:	aspect histograms, from `fullgarf`;
tg_n`.rmf`:	grating RMF files;
tg_n`.arf`:	grating ARF s
tg_n`.pha`:	Type I PHA files (split from `pha2`, for convenience);

in which:

sk:	means the detector subsystem chip, as in `s3`;
tg:	is a grating type, one of `heg`, `meg`, or `leg`;
n:	is the grating order, a signed integer.

Examples in this Function Reference can be useful for seeing how the CIAO tools are invoked by the *S-Lang* wrappers. Some functions call only one tool, some a few. The top-level wrappers will execute a pipeline from start to finish, or part of a pipeline. There is little error-checking, however. If there are problems, the low-level wrappers can be useful for debugging by running one tool at a time.

In the following function definitions, optional function parameters are enclosed in "`[ ]`". In the examples, user input follows the prompt, ISIS, and any output to standard output follows (e.g., `punlearn` is not an input, but is echoed to show the CIAO tools executed).

Some abbreviations or conventions used in function names to distinguish the grating, detector, and mode combinations are:

`lh`:	LETG/HRC
`la`:	LETG/ACIS
`ha`:	HETG/ACIS

`te`: Timed Exposure
`cc`: Continuous-Clocking mode

The organization of the following sections is from high-level to low-level, grouped by instrument or similar functionality.

9 High-level Functions

The high-level functions can be used to set up and run end-to-end processing, from events to spectra, light curves, responses, and summary plots. Other functions load the spectra and responses from the products directory.

There are some functions which set a "state" for non-standard spectral extraction. These are transient so that non-default parameters do not persist over successive extractions. Currently these control the zeroth order detection method and position (via `tgdetect` or `findzo`), the width of sky spatial masks (`tg_create_mask` parameters), and extraction region widths (`tgextract` parameters).

Many of the high-level functions either determine the instrumental configuration from the data at run-time. While low-level functions provide for detailed control of every tool wrapper, the high-level "state" variables support commonly needed extractions without the need for writing low-level scripts.

`s = set_source_detection_info( method[, coord_type, u, v] )` or

`snew = set_source_detection_info( s )`

A "state" function to specify the method for detecting sources, and optionally a coordinate type and the coordinates. This function stores the information internally, and returns a copy of the information in a structure. The valid methods are:

`"tgdetect"` Use the CIAO tool,`tgdetect` (default method)
`"findzo"` Use the ISIS script, `findzo`, which determines the zeroth order's centroid from the intersection of the MEG spectrum and the ACIS frame-shift streak (only good for bright sources, but necessary if the zeroth order is severely distorted by pileup or was excluded from telemetry)
`"none"` Do not attempt to detect the zeroth order. Instead, use the specified coordinates, or if none are given, use target position from the header.

If no other arguments are given, the target position is taken from the event file's header `RA_TARG` and `DEC_TARG` keyword values.

To specify an arbitrary position the optional argument `coord_type` can be either:

`"pixel"` for sky pixel values;
`"radec"` for celestial coordinates in decimal degrees.

If the `coord_type` is given, then the following u, v arguments give the actual coordinate in either pixels or celestial coordinates.

Execution without arguments resets the method to defaults of `tgdetect` and `pixel`.

Example: use `findzo` and a celestial coordinate:

```
isis> s = set_source_detection_info( "findzo", "radec", 123.45678, -23.4567 ) ;
isis> run_pipe( pdir; detect_info = s ) ;
```

Example: use `tgdetect` at the header-specified source location:

```
isis> s = set_source_detection_info( "tgdetect" ) ;
isis> run_pipe( pdir; detect_info = s ) ;
```

In the second form of the usage, the single argument is a structure of the same form that the function returns.

Note: the stored information is transient. After successful source detection, the internally stored method is reset to the default.

`s = get_source_detection_info()`

Get the "state" information controlling source detection. Example:

```
isis> print( get_source_detection_info ) ;
{method="tgdetect",
 coord_type="pixel",
 u=NULL,
 v=NULL}
```

`s = set_tg_create_mask_info( [rzo, wleg] [; arcsec] )` or

`s = set_tg_create_mask_info( [rzo, wheg, wmeg ] [; arcsec] )` or

`snew = set_tg_create_mask_info( s )`

A "state" function to specify the sky region parameters used by `tg_create_mask`. The default units are sky pixels. The parameters are the zeroth order radius, `rzo`, the LEG region full width, `wleg`, or the HEG, MEG region full-widths, `wheg`, and `wmeg`.

An optional qualifier can be given to indicate that units are in imaging arcsec.

This function stores the information internally, and returns a copy of the information in a structure, which can be passed to `run_pipe`.

Example: use narrow sky masks for HETG, 15 arcsec for zeroth order, and 15 arcsec each for HEG and MEG.

```
isis> s = set_tg_create_mask_info( 15, 15, 15; arcsec );
isis> run_pipe( pdir; mask_info = s ) ;
```

In the last usage form, the single argument is a structure of the same form that the function returns.

Execution without arguments echoes the usage and also resets to default parameters. After execution of `tg_create_mask` by high-level functions, the state is reset to the defaults.

`hetgs_use_narrow_masks( flag )`

This is a "state" function to specify that the HETGS sky regions created by `tg_create_mask` should have minimal widths. This is to provide the best HEG flux from about 1.2–1.7 Å, and is useful for very bright and hard sources. It does so at the expense of background regions, but in these cases, background is usually negligible.

If the argument is non-zero, narrow masks are used (25 pixels for HEG and MEG).

The state is transient; after a mask is successfully created (the "`reg1a`" file), the state is reset to the default standard masks.

`s = set_tgextract_info( s_min, s_max[, bd_min, bd_max, bu_min, bu_max] [ ; arcsec] )`
or

`snew = set_tgextract_info( s )`

A "state" function to specify the spectral extraction region parameters used by `tgextract`. The parameters are the minimum and maximum limits for the source spectrum, `s_min` and `s_max`, and optionally the limits for the "background down" (`bd`) and "background up" (`bu`) regions which generally flank the source region. If background limits are omitted, then no background spectra will be extracted.

The default units are grating coordinate degrees. An optional qualifier can be used to indicate that units are in imaging arcsec.

This function stores the information internally, and returns a copy of the information in a structure, which can be passed to `run_pipe`.

Example: use wider than default region for HETG (about -3 to 3 arcsec), and no background:

```
isis> s = set_tgextract_info( -25, 25; arcsec ) ;

isis> run_pipe( pdir; extract_info = s ) ;
```

Execution without arguments echoes the usage and also resets to default parameters. After execution of `tgextract`, the state is reset to the defaults.

In the last usage form, the single argument is a structure of the same form that the function returns.

`s = set_tgre_osort_info( osort_lo, osort_high )` or

`snew = set_tgre_osort_info( s )`

A "state" function to specify the spectral extraction order-sorting limit parameters used by `tg_resolve_events`. The parameters are the minimum and maximum limits for the orders, `osort_lo` and `osort_hi`. Use of these also implies `osipfile = none`.

This function stores the information internally, and returns a copy of the information in a structure, which can be passed to `run_pipe`.

Example: use wider than default region for poor gain correction in CC-mode:

```
isis> s = set_tgre_osort_info( 0.2, 0.5 ) ;
isis> run_pipe( pdir; osort_info = s ) ;
```

Execution without arguments resets to default parameters. After execution of `tg_resolve_events`, the state is reset to the defaults. In the last usage form, the single argument is a structure of the same form that the function returns.

The current setting can be retrieved with

```
isis> s = get_tgre_osort_info ;
```

`run_pipe( pdir[; [detect_info=a][, mask_info=b][, extract_info=c]])`

(NOTE: renamed from `run_cfg`.)

Run the full grating pipeline, from events to spectra, responses, and summary plots, on data in directory `pdir`, writing results into that directory. Determine the configuration from the data in that directory (nominally from an event file).

The optional arguments are structure as returned by `set_*_info` functions described above. If omitted, the default action is to use `tgdetect` at the target location indicated by the data file's header.

The directory must be as downloaded from the Chandra archive and configured with `setup_obsdir` to create the generic file names required by the processing scripts.

Summary plots and tables have the prefix `summary`. See `summary_*.ps` for images of the field, counts spectra, light curves, and other diagnostic plots. See `summary.tbl` for count rates and fluxes, and `summary_fprops.fits` for more detailed spectral properties.

`tg_split_pha_typei( [fpha[, orders[, addkey]]])`

Split a Type II (multi-row) PHA file into single-spectrum files (Type I). By default, split the file `./pha2` into orders +1 and −1 and edit the response files keywords. (Note: `run_pipe()` does this automatically.)

Arguments: zero, 1, 2, or 3 arguments may be given in order:

`fpha:` Type II PHA file.

`orders:` integer array of diffracton (`tg_m`) orders to split.

`addkey:` if non-zero, edit the header keywords, `ANCRFILE` and `RESPFILE`, in the resulting pha files to the default `ARF` and `RMF` products' filenames.

Output `PHA` files are always named: $<$ *grating_type* $>$ _ $<$ *order* $>$ *.pha* and will be written to the input `pha2` file's directory.

Example:

```
isis> tg_split_pha_typei ;
```

Which, if done in an HETG-product directory, will produce files: `heg_-1.pha`, `heg_1.pha`, `meg_-1.pha`, and `meg_1.pha`, with header keywords referring to `heg_-1.arf`, `heg_-1.rmf`, etc.

`tg_run_cleanup( indir, outdir[, rname ] )`

Move the processing products from `indir` to `outdir`, creating the destination directory if necessary. If the summary products have a non-standard prefix, the prefix can be given with the optional parameter.

Note that after this cleanup, processing scripts might no longer find the input data they require, since all inputs are assumed to be in the same directory. Move data when done, if desired, to separate your analysis products from the archival input files.

`tg_run_cleanup_l2( indir, outdir[, rname ] )`

Move only the "Level 2" processing products from `indir` to `outdir`, creating the destination directory if necessary. If the summary products have a non-standard prefix, the prefix can be given with the optional parameter.

This is useful if you want to extract two or more sources from the same field, since it preserves the aspect histogram and intermediate event files for re-use.

`load_set_acis( dir[, rows ] )`

Load a an HETG/ACIS`pha` file and associated ARFs and RMFs. Assumes a file namimg convention: `pha2`, `?eg_<m>.rmf`, `?eg_<m>.arf`. First loads the data (and rows) specified, then uses data table of orders and parts to load responses.

Returns a structure of information on the result. Example

```
isis>  d = load_set_acis( "mydata", [3,4,9,10] ) ;
```

The returned value, `d`, is a structure with the fields: `f`, the file name found; `n`, the array of row indices; `h`, an integer array of histogram indices loaded; `a`, an integer array of ARF indices loaded and assigned; `r`, an integer array of RMF indices loaded and assigned.

`locate_rows( fpha, parts[], orders[] [, srcid] )`

Locate the rows in the named PHA file for the given grating parts and orders, and optionally source ID, if the PHA file happens to contain more than one source's spectra (usually, it doesn't).

If `parts` or `orders` are arrays, then assume we want all listed orders for all parts. For example:

```
isis> fpha = "mydata/pha2" ;
isis> parts = [1,2]; % 1=> HEG, 2=> MEG
isis> orders = [-1,1] ;
isis> r = locate_rows( fpha, parts, rows ) ;
```

means, find the row numbers for `part=1, orders -1,1, part=2, orders -1,1`. For default 12-row files, this results in `r = [3,4,9,10]`.

The ordering of the returned row indices is not specified.

`load_set_hrc( dir, maxord )`

Load an LETG/HRC PHA file and responses from directory `dir` up to `maxord` (absolute value of the order). Uses the default naming scheme as in `load_set_acis`.

Returns a structure of information on the result. Example

```
isis>  d = load_set_hrc( "mydata", 8 ) ;
```

which load orders −8 to 8 and assigns them to the PHA orders. This function also loads and assigns background files (`pha2_bg_-1, pha2_bg_1`) from thi input directory. The returned structure is similar to that for `load_set_acis`, with the addition of the field, `b`, which contains the names of the bacgkround files.

9.1 Generic Detection and Mask Subsidiary Functions

These functions are related to the zeroth order detection and the creation of the spatial mask (made by `tg_create_mask` and used by `tg_resolve_events`).

`pos_info = zodetect( [info],[tgdetect_newpar] )`

Perform source detection if the method is "`tgdetect`" or "`findzo`", store the information, and also return a structure giving source position information.

If the method is `"none"`, simply store the information and also return it. If the method is `"none"` and no coordinates were given, then use the header target coordinates.

Optional arguments may specify the detection information (see `set_source_detection_information`) or parameters for `tgdetect` to overlay the defaults (see `make_newpar`). With no arguments, the stored method (or defaults) are used.

Note that this function has a side-effect: it restores the detection method to the defaults after completion.

The returned value is a structure which contains the fields:

`method:` source detection method (`"tgdetect"`, `"findzo"`, or `"none"`)

`x:` source sky pixel x coordinate;

`y:` source sky pixel y coordinate;

`ra:` celestial coordinate right ascension, in decimal degrees;

`dec:` celestial coordinate, declination, in decimal degrees;

`offaxis:` off-axis angle, from optical axis to source, in arcminutes.

`s = get_source_position_info()`

This is an informational function which retrieves the source position information structure stored by `zodetect()`. This information is used for `tg_create_mask` automatic parameter determination. The return value is the same structure returned by `zodetect`.

`tgdetect_at_targ( [np] )`

Run `tgdetect` using the source position in the event file's header, given by the `RA_TARG` and `DEC_TARG` keywords. The optional argument is for new `tgdetect` parameters, which will overload the defaults (see `make_newpar`).

`tgdetect_at_radec( [np,] ra, dec )`

Run `tgdetect` at the specified celestial coordinates, given in decimal degrees. Parameters for `tgdetect` may be overridden with the optional argument.

`tgdetect_at_xy( [np,] x, y )`

Run `tgdetect` at the specified celestial coordinates, given in sky x, y pixels. Parameters for `tgdetect` may be overridden with the optional argument.

`tg_create_mask_auto( [np,] [fobs, fsrc1a] )`

Run `tg_create_mask` after determination of instrument and source position dependent parameters either from the stored configuration (see `run_init` under Utility Functions, Section 9.4) or from the files referenced. Optionally overload `tg_create_mask` parameters with the new-parameter container, if provided.

This function will determine the zeroth order radius and grating arm widths appropriate for the source's off-axis angle. With no arguments, the instrument (detector) and grating type are assumed to have been cached by an initialization function (`run_init`), and the source position by `zodetect`.

Two reference files may be given explicitly, if desired:

`fobs`: is an observation reference, such as an event file, which has a header containing the instrument and grating types.

`fsrc1a`: is the source table as output by `tgdetect` (a "`src1a`" FITS table), which contains the source coordinates.

The `tg_create_mask` output is a sky spatial region file defining the zeroth and grating arm regions and is named "`reg1a`".

`tg_create_mask_manual_size( rzo, wlheg[, wmeg, [fobs, fsrc1a]] [; arcsec, newpar=np])`

Provides detailed control of `tg_create_mask` region size parameters, and allows them to be specified in arcsec or detector pixels. Assumes that the souce position has either been stored (via `zodetect()`), or can be read from a file (`src1a`, typically).

The `tg_create_mask` output is a sky spatial region file defining the zeroth and grating arm regions and is named "`reg1a`".

Supports the `newpar` container as a way to override other parameters, via a qualifier.

The first two arguments are required and imply that an LETG mask should be created.

Three arguments implies generation of an HETG mask.

Four arguments imply LETG parameters and reference files: `rzo, wleg, fobs, fsrc1a`.

Five arguments imply HETG and reference files: `rzo, wheg, wmeg, fobs, fsrc1a`.

The two reference files are:

`fobs:` is an observation reference, such as an event file, which has a header containing the instrument and grating types.

`fsrc1a:` is the source table as output by `tgdetect` (a "`src1a`" FITS table), which contains the source coordinates.

Also see `tg_create_mask_auto_size` for high-level, from-source-info interface.

`r = tgc_zo_radius( offax, inst, grat )`

Return the zeroth order radius in sky pixels for the given instrument, given `offax`, the off-axis angle in arcminutes, `inst`, the instrument (`acis` or `hrc`), and `grat`, the grating type (either `hetg` or `letg`).

This value can be used in the `tg_create_mask` user parameters (`sX_zero_rad`). The cached values for the configuration and position may be read with `get_source_position_info` and `get_config`.

This function is used by `tg_create_mask_auto`.

`w = tgc_mask_width( offax, inst, grat )`

Return the grating region widths in sky pixels for the given instrument, given `offax`, the off-axis angle in arcminutes, `inst`, the instrument (`acis` or `hrc`), and `grat`, the grating type (either `heg`, `meg` or `leg`). The cached values for the configuration and position may be read with `get_source_position_info` and `get_config`.

This function is used by `tg_create_mask_auto`.

`tg_create_mask_hetgs_minimal( [np,] [fobs, fsrc1a] )`

Run `tg_create_mask` with parameters to create a minimal-width (25 pixels) region file ("`reg1a`") for HETG. This is for bright, hard sources for which the 1.2–1.7 Å HEG flux is important. The default HEG and MEG regions overlap at short wavelengths and event sorting between gratings becomes ambiguous and events are rejected.

A consequence of these narrow masks is that there is no accomodation for background regions (for assessing source confusion as well as background), but in these cases it is not relevant (in general, background for HETGS is extremely low).

9.2 ACIS-specific Subsidiary Functions

`run_acis_1( [exec_flag [ , pdir ] ] )`

Run `acis_process_events` and ACIS event filtering, including `destreak`. The defaults are to excecute verbosely, and to use the current working directory, "`./`". Either or both can be changed with the optional parameters.

The input file is `evt0`, and the final output is `evt1`. Several intermediate event files are written, and not removed.

Example: Use the do-not-execute flag, and give the current directory:

```
isis> run_acis_1(0, ".");
% Mode: 0 ( DO NOT EXECUTE )

dmmakepar \
  output='evt0.par' \
  mode='hl' \
  input='evt0[events]' \
  clobber='yes'

punlearn acis_process_events

acis_process_events \
  apply_cti='yes' \
  apply_tgain='yes' \
  mode='hl' \
  verbose='0' \
  gainfile='CALDB' \
  outfile='evt1_0' \
  acaofffile='@asol.list' \
  eventdef=')stdlev1' \
  obsfile='NONE' \
  infile='evt0' \
  rand_pix_size='0' \
  ctifile='CALDB' \
  badpixfile='bpix1' \
  clobber='yes'

punlearn destreak

dmcopy \
  mode='hl' \
  outfile='evt1_1' \
  opt='all' \
  infile='evt1_0[events][grade=0,2,3,4,6,status=0,energy<12000]' \
  clobber='yes'
dmcopy \
  mode='hl' \
  outfile='evt1_2' \
  opt='all' \
  infile='evt1_1[events][@flt1]' \
  clobber='yes'
$ASCDS_INSTALL/bin/destreak \
  filter='yes' \
  mode='hl' \
  outfile='evt1' \
  infile='evt1_2' \
  clobber='yes'
```

`run_tg_acis_1a( [exec_flag [, pdir]] )`

Run the grating part of the pipeline from zeroth order source detection through response generation. Uses the stored state for the detection method and sky masks.

`run_tg_acis( [exec_flag [, pdir]] )`

Run the grating pipeline from `acis_process_events` through response generation. Runs `run_acis_1` and `run_tg_acis_1a`.

9.2.1 A Detailed Example of a Customized Processing Function

`run_tg_acis_custom_1a( [exec_flag[,pdir]] )` This is an example of a customized grating processing function, which defines narrow HEG and MEG spatial regions for a source in a crowded field. It is worth listing the entire function definition here for tutorial purposes:

```
% Example of customized extraction.
% Modify tg_create_mask parameters:
% Extract from narrow mask (crowded field, e.g., obsid 4587):
%
   define run_tg_acis_custom_1a( )   % ( [exec_flag[,pdir]] )
   {
     % get any optional arguments, which are the exec_flag (0, 1, or
     % 2), and the data directory.  run_init() will chdir to the data
     % directory, read the configuration, save it in memory, and
     % create any necessary setup parameter files.

       variable args = __pop_args( _NARGS ) ;
       run_init( __push_args( args ) ) ;

       variable x = -1.0, y = -1.0, c = -1.0 ; % placeholder, invalid values
       ( x, y, c ) = read_src1a_pos;  % Use the existing tgdetect output

       % shrink regions - for close sources: (e.g., obsid 4587)
       %
       variable rzo = 28, wheg = 8, wmeg = 8 ;    % CUSTOMIZATION
       tg_create_mask( x, y, rzo, wheg, wmeg) ;   % CUSTOMIZATION

       tg_resolve_events ;
```

```
        tgextract ;

        update_std_file_zomethod( "TGDETECT" ); % record zo method in pha2 header

        lightcurve ;

        variable orders = [ -1, 1 ] ;
        make_responses( orders ) ;

        run_close ;   % goes back to starting directory.
    }
```

9.3 HRC-specific Subsidiary Functions

`run_hrc_1( [exec_flag[, pdir]] )`

Run `hrc_process_events`. The defaults are to excecute verbosely, and to use the current working directory, "`./`". Either or both can be changed with the optional parameters.

The input file is `evt0`, and the final output is `evt1`.

Example: Use the do-not-execute flag, and give a directory:

```
isis> run_hrc_1( 0, "Data/obs_6441" ) ;
% Mode: 0 ( DO NOT EXECUTE )

dmmakepar \
  output='evt0.par' \
  mode='hl' \
  input='evt0[events]' \
  clobber='yes'

punlearn hrc_process_events

hrc_process_events \
  mode='hl' \
  verbose='0' \
  gainfile='CALDB' \
  outfile='evt1' \
  acaofffile='@asol.list' \
  eventdef=')stdlev1' \
  obsfile='none' \
  infile='evt0[events][pha=0:254,status=xxxxxx00xxxx0xxx0000x000x00000xx,@flt1]'\
  rand_pix_size='0' \
  badpixfile='bpix1' \
  instrume='hrc-s' \
  clobber='yes'
```

`run_tg_hrc( [exec_flag[, pdir]] )`

Run the grating pipeline from `hrc_process_events` through response generation. Runs `run_hrc_1` and `run_tg_hrc_1a`.

`run_tg_hrc_1a( [exec_flag[, pdir]] )`

Run the grating portion of the pipeline starting with the output of `hrc_process_events`, and ending with grating responses.

9.4 Utility Functions

9.4.1 Process Control

`run_init( [exec_flag[, pdir]] )`

Sets the execution flag, if given, or uses the default of 2 for `EXECUTE_VERBOSE`). Use 0 for no-execution (for low-level CIAO wrappers).

Changes directories to `pdir`, if given (default is ".").

Sets up parameter file path to `./param` and creates the directory if it doesn't exist.

Reads the configuration from `pdir/evt0` (a symbolic link to the archive `evt1` file), and "pretty-prints" the configuration to `pdir/obs_config.txt`.

Makes an "obspar" file used by some CIAO tools, `pdir/evt0.par`.

`run_close:`

Closes out a run, initialized by `run_init`. Currently, this is nothing more than a `popd` to return to the starting directory.

`pushd( pdir ):`

Push-directory: push the current directory onto a list, and change the working directory to pdir.

`popd:`

Pop the current directory off the stored list, and change to the previous directory.

`set_punlearn( flag )` Set a flag which controls whether the CIAO `paramio` tool, `punlearn`, is executed before each CIAO tool is run. If `flag` = 0, don't run `punlearn`.

`maybe_punlearn( toolname )` Conditionally run `punlearn` depending on the state of the flag.

Example:

```
isis> set_punlearn(0);
%% punlearn DISABLED
isis> maybe_punlearn("acis_process_events");
isis> set_punlearn(1);
%% punlearn ENABLED
isis> maybe_punlearn("acis_process_events");

punlearn acis_process_events
```

`newpar = make_newpar( ; param1=val1[, param2=val2[, ...]]  );`

Use *S-Lang* qualifiers (the comma-delimited *name* = *value* pairs following the semicolon) to create a parameter variable used to overlay default parameters in CIAO tool wrapper functions.

This is primarily to support rather general customization of processing parameters for special cases.

Example: Set the verbosity level in `acis_process_events` to 5, and turn off CTI correction:

```
np = make_newpar( ; verbose=1, apply_cti="no" ) ;
set_exec( 0 ) ;
acis_process_events( np ) ;
```

The returned value is an associative array which contains data of type `Any_Type`. The above example could also be constructed explicitly via:

```
np = Assoc_Type[ Any_Type ] ;
np[ "verbose" ] = 1 ;
np[ "apply_cti" ] = "no" ;
acis_process_events( np ) ;
```

This function must be used with care, if altering input/output file names, since some of the tool-wrappers construct filenames, or expect particular input filenames.

9.4.2 Coordinate Transformations

`(x,y) = radec_to_xy( f_wcs_ref, ra_deg, dec_deg )`

Convert a celestial coordinate, given in decimal degrees, to the sky pixel position using the World Coordinate System (WCS) specified in a reference file, `f_wcs_ref`, such as an event file.

`(x,y) = get_targ_xy( f_wcs_ref )`

Get the sky pixel coordinates for the target position given in the header of the referenced file, using the WCS from the same header.

`(ra, dec) = xy_to_radec( f_wcs_ref, x, y )`

Convert the sky pixel coordinates, x, y, to decimal right ascension and declination, using the WCS in the header of the referenced file.

`rad = arclen( ra1, dec1, ra2, dec2 )`

Compute the great circle angle between two celestial coordinates, each given in decimal degrees. The result is given in radians.

`arcmin = xy_to_offaxis( f_wcs_ref, x, y )`

Compute the off-axis angle for sky pixel coordinates, (x, y), using the WCS and axial coordinates in the header of the referenced file. The result is given in arcminutes.

`arcmin = radec_to_offaxis( f_wcs_ref, ra, dec )`

Compute the off-axis angle for sky celestial coordinates, (ra, dec), given in decimal degrees, using the WCS and axial coordinate in the header of the referenced file. The result is given in arcminutes.

10 Functions to Produce Quick-Look Summary Plots

A collection of functions will generate graphical output (postscript files) and a few FITS files to serve for quick-look and review of the data and to validate the processing. The high-level functions run several subsidiary tasks which each generate one kind of plot or table. There are few or no options for customization. That can be done in off-line analysis.

`tg_summary( pdir, rname )`

Top-level, creates several summary plots and tables. Reads data products in pdir and determines instrumental configuration for calling of appropriate subsidiary functions.

Write data with prefix, `rname`, which may include a path to a directory (which should exist).

The plots incude images of the field in sky coordinates, field images in spectral coordinates, counts spectra, flux spectra (if ACIS), light curves, and order-sorting plots (if ACIS).

In addition, there are some FITS files and ASCII tables created. The suffix is a mnemonic for the type of plot:

`_flux_overview.ps`	Flux spectrum
`_spc.ps`	Counts spectrum
`_spf.ps`	Flux spectrum
`_lc.ps`	Light curve
`_im-a.ps`	Source region detail images
`_im-b.ps`	Sky field image
`_imsp.ps`	Dispersion coordinates field image

If ACIS, then order-sorting products are made:

`_im_osum.ps`	Order-sorting image
`_heg_123.fits`	HEG order sorting FITS, image, as order sorted by `tg_resolve_events`
`_heg_all.fits`	HEG order sorting FITS image, showing wide order regions
`_meg_123.fits`	MEG order sorting image, as order sorted.
`_meg_all.fits`	MEG order sorting image, wide order regions
`_leg_123.fits`	LEG/ACIS order sorting image, as order-sorted
`_leg_all.fits`	LEG/ACIS order sorting image, wide order regions
`_fprops.fits`	Rates and fluxes in bands.
`.tbl`	ASCII table of configuration, summary rates and fluxes.

`plot_flux_overview( h )`

Make an overview plot of flux density vs energy, combining first orders and binned to some minimum signal to noise ratio and channel grouping. The argument, *h*, give the indices of the spectral histograms, which must have assigned responses. The plot is to the currently open device.

In the case of HETG, the HEG and MEG first orders are combined on an MEG grid.

In the case of LETGS, the flux is very approximate because of unresolvable overlapping orders; the background-subtracted counts are treated as if they were all first order.

`rd_data_plot_flux_overview( pdir[, rname[, devopt ] )]`

Read the counts spectrum and responses from standard products in the named directory, `pdir`, and make a flux density vs. energy overview plot. The output files can have a filename prefix specified by `rname`, whose default is `./summary`. The suffix `_flux_overview` will be appended, along with a device name, whose devault is `.ps`.

If a device other than postscript is desired, the `devopt` option can be used to give any valid `pgplot` device. The syntax is `.<ext>/<dev>`, where `.<ext>` is an arbitrary filename extension, and `devopt` is the `pgplot` device name. Examples are `.ps/vcps`, `.gif/gif`, and `.png/png`. The `devopt` string mus begin with a "`.`" and contain a "`/`"

Plotting is done by `plot_flux_overview`

Example: `isis> rd_data_plot_flux_overview( "obs_1234", "test");` to create file: `obs_1234/test_flux_overview.ps`.

Example: `isis> rd_data_plot_flux_overview( "obs_1234", "test", ".gif/gif");` to create file: `obs_1234/test_flux_overview.gif`.

Example: Make summary plots and tables for products found in `Data/obs_3`. Write summary products to `./summary_*.*`.

```
tg_summary( "Data/obs_3", "./summary" );
```

10.1 Subsidiary HETG/ACIS Functions

`hetgs_summary( pdir, rname )`

Do all the summary plots and flux properties for an HETG/ACIS observation in directory `pdir`, and give the summary products the root name, `rname`.

`hetgs_disp_summary( [pdir[, rootname]] )`

Create the dispersion-coordinate field image. The default directory is "`./`", which must contain the files `evt2` and `pha2`.

`hetgs_field_summary( [pdir[, rootname]] )`

Make the field images in sky coordinates centered on the zeroth order, zooming out by factors of 2 several times to show successively larger areas.

`hetgs_hires_flux_summary( pdir, rootname )`

Generate a plot of the fluxed spectrum from the `pha2` and response files found in directory `pdir`, giving it the prefix, `rootname`.

`hetgs_lc_plot( pdir, rootname )`

Plot the count-rate light curve found in file `pdir/lc`, and give it the path/prefix, `rootname`. If a background light curve file exists (`lc_bg`), include it in the plot.

`hetgs_osum_img( pdir, rname )`

Make the order-sorting images from the product files in `pdir`. The order-sorting images show the distribution of events in real-valued order (as determined from the CCD energy) against grating wavelength. The two part labeled "123" shows the as-order-sorted (by `tg_resolve_events`) events, and the "all" images show all events, in order to assess any inadvertant clipping due to calibration uncertainties. This function creates both FITS image files and postscript plots.

`hetgs_spec_cts_summary( pdir, rootname )`

Make a plot of the counts vs wavelength.

10.2 Subsidiary LETG/ACIS Functions

The following are analagous to the HETGS counterparts:

```
la_summary( pdir, rname )
la_disp_summary( [pdir[, rootname]] )
la_field_summary( [pdir[, rootname]] )
la_hires_flux_summary( pdir, rootname )
la_lc_plot( pdir, rootname )
la_osum_img( pdir, rname )
la_spec_cts_summary( pdir, rootname )
```

10.3 Subsidiary LETG/HRC Functions:

The following are analagous to the HETGS counterparts. With HRC-S, there is no order-sorting, and so there can also be no model-independent flux, so those counterparts are absent.

```
lh_summary( pdir, rname )
lh_disp_summary( [pdir[, rootname]] )
lh_field_summary( [pdir[, rootname]] )
lh_lc_plot( pdir, rootname )
lh_spec_cts_summary( pdir, rootname )
```

10.4 Flux Properties

`hetgs_fprops( pdir, rname[,ftbl ] )`

Compute some flux properties for the HETGS products (`pha2`, ARF s, and RMF s) found in directory `pdir`, assigning the rootname, `rname`, for the combined positive and negative first orders. The properties are rates and fluxes in wavelength bands given in the default table, or in a table referred to by the optional argument. If a custom table is provided, it should have 3 columns which give the minimum wavelength, the maximum wavelength, and a label.

The output is a FITS binary table.

`la_fprops( pdir, rname[,ftbl ] )`

This is analogous to `hetgs_fprops`, but tailored for LETG/ACIS.

`lh_fprops( pdir, rname[,ftbl ] )`

This is analogous to `hetgs_fprops`, with modifications for LETG/HRC-S (there is no flux computed, since the overlapping orders preclude flux determination without a model spectrum).

`pprint_fprops( ftbl[, brief][, file] )`

"Pretty-print" a flux-properties table stored in FITS binary table file, `ftbl`. If argument `brief` is 1, then print a brief version. If a string argument, `file` is specified, then print to an ASCII file; otherwise, to standard output.

`( r, sigma_r ) = get_zo_rate( fevt2 )`

Get the zeroth order rate and statistical uncertainty from the event file specified by `fevt2`. This is used by the above flux-properties functions.

11 Library Functions, for Low-Level Use

These functions comprise a library of wrappers for CIAO data processing commands. Most only set a few key parameters after execution of `punlearn` to initialize default parameters.

11.1 Instrumental Configuration, Processing Control

`c = read_config( obsfile )`

Read the instrumental configuration details (detector, grating, mode) and observational information (object, exposure time) from the "obsfile", a FITS file which has the information about the observation. This is typically an event file.

This returns a structure containing the information.

This is important to execute before using the higher-level functions which rely on a stored configuration in order to choose calls to detector, grating, or mode-specific functions.

`c = get_config()`

Retrieve the current configuration information from memory, as stored by `read_config`.

`pprint_config( c [, file] )`

"Pretty-print" the configuration information in the configuration structure, `c`. If a file name is given, write to the file instead of standard output.

Example:

```
isis> c = read_config( "Data/obs_5/evt0" );
isis> pprint_config( c ) ;
     OBSID:  5
    OBJECT:  TW HYA
  DATE_OBS:  2000-07-18T10:00:54
   RA_TARG:  165.467
  DEC_TARG:  -34.7044
  ROLL_NOM:  226.643
  INSTRUME:  acis
    DETNAM:  acis-456789
   GRATING:  hetg
  EXPOSURE:  47756.2
   TIMEDEL:  3.24104
  READMODE:  timed
  DATAMODE:  vfaint
```

`set_exec( [flag] )`

Set a flag to control execution of CIAO tools. The flags values meanings are:

0	do not execute
1	execute silently
2	execute verbosely (the default)

This is primarily a debugging utility used to echo CIAO commands and parameters without execution. But it is not strictly guaranteed to not perform some file i/o. Some functions do ad hoc i/o via *S-Lang* cfitsio to obtain information from a file.

11.2 ACIS/HRC Level 1 Event Processing and Filtering

`acis_process_events( [newpar] )`

Conditionally run `acis_process_events`, using the stored instrumental configuration to determine whether the mode was continous clocking (CC) or timed exposure (TE) from the stored configuration.

Input/output are to/from the current directory. The input file is `evt0` (a symbolic link to the archive `evt1` file), and the output is named `evt1_0`, a temporary file to be filtered.

Requires configuration to have been established with `read_config()`.

Respects the current exec flag.

Example:

```
isis> run_init(0,pdir);
% Mode: 0 ( DO NOT EXECUTE )

dmmakepar \
  output='evt0.par' \
  mode='hl' \
  input='evt0[events]' \
  clobber='yes'

isis> acis_process_events;

punlearn acis_process_events

acis_process_events \
  apply_cti='yes' \
  apply_tgain='yes' \
  mode='hl' \
  verbose='0' \
  gainfile='CALDB' \
  outfile='evt1_0' \
  acaofffile='@asol.list' \
  eventdef=')stdlev1' \
  obsfile='NONE' \
  infile='evt0' \
  rand_pix_size='0' \
  ctifile='CALDB' \
  badpixfile='bpix1' \
  clobber='yes'
```

`acis_process_events_cc( [ newpar ] )`

Run `acis_process_events` with parameters chosen for continuous-clocking (CC) mode. (Does not rely on the stored configuration.)

`acis_process_events_te( [ newpar ] )`

Run `acis_process_events` with parameters chosen for time-exposure (TE) mode. (Does not rely on the stored configuration.)

`acis_evt_filter_1( [ newpar ] )`

Filter the ACIS temporary event file (`evt1_0`) on grade, status, and energy, using dmcopy. The filter is defined as

```
 grade=0,2,3,4,6,status=0,energy<12000
```

Write to a new temporary file, `evt1_1`.

If the optional newpar argument is given, it must be a container as defined by `make_newpar()`.

`acis_evt_filter_2( [ newpar ] )`

Filter the ACIS temporary event file, `evt1_1` (output by `acis_evt_filter_1`) on the GTI tables contained in the file, `flt1`, using `dmcopy`.

Write to a new temporary file, `evt1`.

If the optional `newpar` argument is given, it must be a container as defined by `make_newpar()`.

`acis_evt_filter()`

Run `acis_evt_filter_1` and `acis_evt_filter_2` in succession.

Note that parameter modification is not supported. To alter filter paramaters, run the individual functions.

`destreak( [newpar ] )`

Run the CIAO `destreak` filter on the ACIS `evt1` file, and write to file `evt1_ds`.

If the optional `newpar` argument is given, it must be a container as defined by `make_newpar`.

`acis_evt_filter_ds()`

Run the three ACIS event filters in succession: `acis_evt_filter_1`, `acis_evt_filter_2`, and `destreak`. The input is the `evt1_0` file output by `acis_process_events`. Intermediate files are `evt1_1`, `evt1_2`, and the `destreak` output, `evt1`.

The file, `evt1`, is the final product of `acis_process_events` and filtering, and is to be the input to `tg_resolve_events`.

Note that the `newpar` argument is not supported. If customization is required, use the subsidiary filtering functions.

`make_acis_filter_list( [fref[, fout] ])`

Write a filter file as an alternative function for ACIS event filtering, instead of the filters 1 and 2 above. This replaces the detail of the GTI (Good Time Interval) tables with one time interval spanning the observation. This can be useful for CC-mode observations with thousands of GTI intervals which can take a very long time to be processed by `dmcopy`. In principle, there should be no difference in the product in this case since no events are detected in the time gaps to be filtered, which are largely frame-drops.

The first optional argument is the reference file for reading the `TSTART` and `TSTOP` keywords, such as `evt0` (the default).

The second optional argument is the output table. The default is `acis_default_filter.list`.

The filter can be applied with the function, `acis_evt_filter_list()`.

Example:

```
isis> make_acis_filter_list;

isis> !cat acis_default_filter.list
     time=80301654.607780:80351510.609629,
     grade=0,2:4,6,
     status=0,
     energy<12000

isis> acis_evt_filter_list();
     dmcopy \
       mode='hl' \
       outfile='evt1' \
       opt='all' \
       infile='evt1_0[events][@acis_default_filter.list]' \
       clobber='yes'
```

`acis_evt_filter_list( [newpar] )`

Apply the event filter in an "at-file" using `dmcopy`. The input is `evt1_0` which was output by `acis_process_events`. The output is `evt1`.

This is an alternative to the other filters, `acis_evt_filter_1`, `acis_evt_filter_2`.

See `make_acis_filter_list` for an example.

`hrc_process_events( [newpar] )`

Run `hrc_process_events` on the input file, `evt0`, applying the standard filter (pre-CIAO 4.2)

`pha=0:254,status=xxxxxx00xxxx0xxx0000x000x00000xx,@flt1`

and writing the output file, `evt1`.

If the optional `newpar` argument is given, it must be a container as defined by `make_newpar`.

Example:

```
isis> hrc_process_events;

punlearn hrc_process_events

hrc_process_events \
  mode='hl' \
  verbose='0' \
  gainfile='CALDB' \
  outfile='evt1' \
  acaofffile='@asol.list' \
  eventdef=')stdlev1' \
  obsfile='none' \
  infile='evt0[events][pha=0:254,status=xxxxxx00xxxx0xxx0000x000x00000xx,@flt1]' \
  rand_pix_size='0' \
  badpixfile='bpix1' \
  instrume='hrc-s' \
  clobber='yes'
```

Note: For CIAO 4.2, support has been added for the HRC-S time-dependent gain correction (as of `tgcat.sl` version 1.5.0). This is done automatically by `hrc_process_events` and `tg_resolve_events` if the proper CALDB and CIAO 4.2 are detected. In this case, the event filter applied is:

```
infile='evt0[events][rawy=:16383,17070:32200,32980:, \
           status=xxxxxx00xxxx0xxx0000x000x00000xx,@flt1]'
```

A corresponding filter is also applied by `tgextract`:

```
infile=evt2[(tg_mlam,pi)=\
   region($CALDB/.../tgpimask2/letgD1999-07-22pireg_tgmap_N0001.fits)]
```

11.3 Grating Events Processing and Spectrum Binning

`tgdetect( [ newpar ] )`

Run `tgdetect`, the CIAO program which detects and centroids the zeroth order using `celldetect`. The input is the event file, `evt1`, and the output is the source file, `src1a`, whose source position is used by `tg_create_mask`.

The default search position is a box around the nominal source position as determined by the spacecraft pointing and the target coordinates. See the CIAO `ahelp` pages for details.

Example: Use all defaults:

```
isis> tgdetect;

punlearn tgdetect

tgdetect \
  zo_pos_x='default' \
  mode='hl' \
  verbose='0' \
  fixedcell='6' \
  outfile='src1a' \
  zo_sz_filt_x='default' \
  zo_sz_filt_y='default' \
  infile='evt1' \
  zo_pos_y='default' \
  clobber='yes'
```

Example: Override the default source position with an a priori determined coordinate:

```
isis> np = make_newpar( ; zo_pos_x = 3900, zo_pos_y = 4300 );
isis> tgdetect( np ) ;

punlearn tgdetect

tgdetect \
  zo_pos_x='3900' \
  mode='hl' \
  verbose='0' \
  fixedcell='6' \
  outfile='src1a' \
  zo_sz_filt_x='default' \
  zo_sz_filt_y='default' \
  infile='evt1' \
  zo_pos_y='4300' \
  clobber='yes'
```

`tg_create_mask( [newpar,][ x, y[, zorad, wheg[, wmeg] ] ] )`

Run `tg_create_mask`, the CIAO program which writes a region file, `reg1a`, defining the spatial filters for the grating arms and zero order.

`Tg_create_mask` has many parameters to support automatic use and to provide manual customization. The output is the `reg1a` file which is the spatial mask defining the zeroth order and grating arm regions (HEG and MEG if HETG, or LEG if LETG).

The default input and output files are `evt1` and `reg1a`, respectively, and the default `input_pos_table` is `src1a` (all in the current directory).

The options are complicated, allowing from zero to six arguments, as follows.

`tg_create_mask()`

Use parameter defaults, determined by the `evt1` file header for the grating type, using the `src1a` file for the source position. Nominal widths are used for the grating arm regions. These defaults are suitable for single point sources near the nominal aim point.

`tg_create_mask( x, y )`
: Use the (x, y) values for the zeroth order centroid.

`tg_create_mask( x, y, rzo, wheg, wmeg )`
: Specify the source position as well as the radius of the zeroth order region, `rzo`, and the widths of the HEG and MEG regions, `wheg` and `wmeg`, respectively (all given in sky pixels). This is useful for crowded fields to minimize MEG-HEG region overlap area, or for very hard spectra, to minimize HEG-MEG overlap at the shortest wavelengths.

`tg_create_mask( x, y, rzo, wleg )`
: As above, but for LETG which requires only one grating region width.

`tg_create_mask( newpar )`
: Use the arguments given by the `newpar` container to override default parameters.

`tg_create_mask( newpar, x, y )`
: Use the arguments given by the `newpar` container to override default parameters, but use the given x and y location for the zeroth order centroid.

`tg_create_mask( newpar, x, y, rzo, wheg, wmeg )`
: Use the arguments given by the `newpar` container to override default parameters, but use the given x and y location for the zeroth order centroid, and the radius and width parameters for the zeroth order and HETG arms.

Example: Specify a zeroth order centroid with $(x, y) = (4100, 4200)$, a zeroth order region radius of 30 pixels, and HEG and MEG widths of 20 pixels. Note that `use_user_pars='yes'`, which means that the `input_pos_table` value will be ignored:

```
isis> tg_create_mask(4100, 4200, 30, 20, 20);

punlearn tg_create_mask

tg_create_mask \
  sA_width_heg='20' \
  mode='hl' \
  sA_width_meg='20' \
  verbose='0' \
  sA_zero_y='4200' \
  outfile='reg1a' \
  grating_obs='header_value' \
  detector='header_value' \
  use_user_pars='yes' \
  input_pos_tab='src1a' \
  infile='evt1' \
  sA_zero_x='4100' \
  clobber='yes' \
  sA_zero_rad='30'
```

Example: Use the defaults:

```
isis> tg_create_mask;

punlearn tg_create_mask

tg_create_mask \
  mode='hl' \
  verbose='0' \
  outfile='reg1a' \
  grating_obs='header_value' \
  detector='header_value' \
  use_user_pars='no' \
  input_pos_tab='src1a' \
  infile='evt1' \
  clobber='yes'
```

`tg_resolve_events( [ newpar ] )`

Run the CIAO tool, **`tg_resolve_events`**, which computes grating coordinates for the source defined by region **`reg1a`**, for the instrumental configuration currently stored in memory (via **`read_config`**). The relevant configuration values are the readmode (continuous or timed), and the instrument (ACIS or HRC). This calls one of the three subsidiary wrappers (**`tg_resolve_events_cc`**, **`tg_resolve_events_lh`**, or **`tg_resolve_events_te`**).

Use the arguments given by the optional **`newpar`** container to override default parameters.

The input files are **`evt1`**, **`reg1a`**, and the aspect solution list file, **`asol.list`**; the output is **`evt2`**.

Example:

```
isis> tg_resolve_events;

punlearn tg_resolve_events

tg_resolve_events \
  mode='hl' \
  verbose='0' \
  outfile='evt2' \
  acaofffile='@asol.list' \
  regionfile='reg1a' \
  infile='evt1' \
  osipfile='CALDB' \
  clobber='yes'
```

`tg_resolve_events_cc( [ newpar ] )`

Run **`tg_resolve_events`** with parameters for CC-mode. Other parameters may be overridden with the optional container, **`newpar`**.

Example: Note the specification of the `osort_hi`, `osort_lo`, and `osipfile`:

```
isis> tg_resolve_events_cc;

punlearn tg_resolve_events

tg_resolve_events \
  mode='hl' \
  verbose='0' \
  outfile='evt2' \
  acaofffile='@asol.list' \
  osort_hi='0.3' \
  regionfile='reg1a' \
  osort_lo='0.2' \
  infile='evt1' \
  osipfile='none' \
  clobber='yes'
```

`tg_resolve_events_lh( [ newpar ] )`

Run `tg_resolve_events` with parameters for LETG/HRC-S. Other parameters may be overridden with the optional container, `newpar`.

Example: Note the choice of eventdef:

```
isis> tg_resolve_events_lh;

punlearn tg_resolve_events

tg_resolve_events \
  mode='hl' \
  verbose='0' \
  outfile='evt2' \
  acaofffile='@asol.list' \
  eventdef=')stdlev1_HRC' \
  regionfile='reg1a' \
  infile='evt1' \
  osipfile='none' \
  clobber='yes'
```

`tg_resolve_events_te( [ newpar ] )`

Run `tg_resolve_events` with parameters for ACIS in timed-exposure mode. Other parameters may be overridden with the optional container, `newpar`.

Example:

```
isis> tg_resolve_events_te;

punlearn tg_resolve_events

tg_resolve_events \
  mode='hl' \
  verbose='0' \
  outfile='evt2' \
  acaofffile='@asol.list' \
  regionfile='reg1a' \
  infile='evt1' \
  osipfile='CALDB' \
  clobber='yes'
```

`tgextract( [ newpar ], [ orders ] )`

Bin the events in file `evt2` into a spectrum, written to file `pha2`. Determine the instrument-dependent parameters from the stored configuration information (either ACIS or HRC-S).

Either one, both, or none of the optional arguments may be given. Newpar will overlay default parameters with those in the container. The orders parameter should be an array of integer spectral orders to bin. The default for ACIS is $[-3, -2, -1, 1, 2, 3]$, and for HRC, [-1,1].

Example: Use default for the current instrument:

```
isis> tgextract;

punlearn tgextract

tgextract \
  tg_order_list='default' \
  mode='hl' \
  verbose='0' \
  outfile='pha2' \
  infile='evt2' \
  extract_background='yes' \
  inregion_file='none' \
  clobber='yes'
```

Example: Turn off the background extraction and specify only first orders:

```
isis> np = make_newpar( ; extract_background="no");
isis> tgextract( np, [-1,1] );

punlearn tgextract

tgextract \
  tg_order_list='-1,1' \
  mode='hl' \
  verbose='0' \
  outfile='pha2' \
  infile='evt2' \
  extract_background='no' \
  inregion_file='none' \
  clobber='yes'
```

See also: `make_newpar, read_config, tg_resolve_events`

`tgextract_acis( [ newpar ], [ orders ] )` Bin the spectrum, but do not use the stored configuration; use default parameters as for ACIS.

Example:

```
isis> tgextract_acis;

punlearn tgextract

tgextract \
  tg_order_list='default' \
  mode='hl' \
  verbose='0' \
  outfile='pha2' \
  infile='evt2' \
  extract_background='yes' \
  inregion_file='none' \
  clobber='yes'
```

`tgextract_lh( [ newpar ], [ orders ] )`

Bin the spectrum using default parameters suitable for LETG/HRC-S (“`lh`”). (Does not use the stored configuration.) The primary difference from ACIS is the specification of the input file filter, used to reduce HRC-S background.

Example:

```
isis> tgextract_lh;

punlearn tgextract

tgextract \
  tg_order_list='default' \
  mode='hl' \
  verbose='0' \
  outfile='pha2' \
  infile='evt2[(tg_lam,pi)=region($CALDB/.../tgmask2/letgD1999-07-22pireg075_N0001.fits)]' \
  extract_background='yes' \
  inregion_file='CALDB' \
  clobber='yes'
```

Note: for CIAO 4.2 and associated CALDB, HRC-S processing provides a time-dependent gain correction. This is supported by default if CIAO 4.2 and the proper CALDB are detected (since `tgcat.sl` version 1.5.0). In that case, the event filter is

```
  infile=evt2[(tg_mlam,pi)= \
     region($CALDB/.../tgpimask2/letgD1999-07-22pireg_tgmap_N0001.fits)]
```

`tg_bkg_lh( [newpar[ [infile[,outfile]]] ])`

Bin the background spectrum for LETG/HRC-S data. This runs the CIAO contributed script, **`tg_bkg`**, then the CIAO tool, **`dmtype2split`** to copy the individual orders from the two-row file, **`pha2_bg`**, to files with one order each, **`pha2_bg_-1`** and **`pha2_bg_1`**, which can be more convenient for analysis.

The default input file is `./pha2`, and default outputs are `./pha2_bg`, `./pha2_bg_-1`, and `./pha2_bg_1`. Optional parameters allow explicit names to be given.

The `newpar` argument only allows the "rows" parameter to be overridden in `dmtype2split`. The default for LETG/hrcs is two rows, the maximum, since there is no order-sorting capability.

Example:

```
isis> tg_bkg_lh ;

punlearn dmtype2split

pset dmtcalc \
  mode='hl' \
  clobber='yes'

tg_bkg pha2 pha2_bg

dmtype2split \
  mode='hl' \
  verbose='0' \
  outfile='pha2_bg_-1[SPECTRUM],pha2_bg_1[SPECTRUM]' \
  infile='pha2_bg' \
  clobber='yes'
```

`(x0, y0) = c_findzo()`

`Findzo` is an ISIS package which determines the zeroth order centroid by fitting the intersection of a grating arm (typically MEG for HETG) with the ACIS frame-shift streak of the zeroth order. This method is sometimes necessary for blocked zeroth order observations, or for very bright sources in which the zeroth order is cratered due to on-board rejection of bad grades.

This function reads the grating type from the evt1 file header, then run findzo using that type, which will be either MEG (if HETG), or LEG (if LETG).

It returns the zeroth order centroid, $x0, y0$, and writes a plot of the solution in file `findzo.ps`.

It is usually not necessary to run this function explicitly; it is called by other functions.

`Findzo` is independently documented and maintained. Further information can be found at

`http://space.mit.edu/cxc/analysis/findzo`

`r = est_zo_rate( ddir )` A utility function which estimates the maximum zeroth order count rate per frame per 3×3 pixel detection cell. This reads the zeroth order position from the `reg1a` file, blocks a 12×12 pixel region around the that position by 3, and returns the maximum rate scaled by the frametime; that is, the maximum counts/frame in a 3×3 region.

The argument, `ddir`, is the directory in which to find files `evt0` and `reg1a`.

This can be useful for estimating pileup in some count rate regimes.

This is currently not used in the *TGCat* production.

`( x, y, c ) = read_src1a_pos( [fsrc1a] )`

Read the sky x, y position of the zeroth order from a `src1a` file (the output of `tgdetect`). The default input is `src1a` in the current directory. Return the sky pixel position, x, y, and the net counts, c.

11.4 Responses:

`make_responses( orders[, mkgrmf_par[, mkgarf_par] ] ] )`

Make responses (`asphist`, RMF S, ARF S) using the stored instrumental configuration determined from the file `./pha2`.

This is the top-level function for making a full set of responses, including ancillary input files such as aspect histograms.

`orders` is an integer array of diffraction orders.

The next two optional parameters can be used to override default parameters for `mkgrmf` or `mkgarf`. To specify `mkgarf_par` but not `mkgrmf_par`, use `NULL` for `mkgrmf_par`. See `make_newpar` for details on constructing a parameter container variable.

`make_responses` uses the files in current directory, `evt0`, `pha2`, `reg1a`, to obtain configuration information. It writes output to the current directory.
Example (with very abbreviated output):

```
isis> make_responses( [-1,1] );

 punlearn ardlib
 punlearn asphist
 punlearn mkgrmf
 punlearn mkgarf
 punlearn dmarfadd

 asphist \
   dtffile='evt2' \
   mode='hl' \
   res_xy='0.5' \
   verbose='0' \
   evtfile='evt2[CCD_ID=4]' \
   outfile='s0.asphist' \
   max_bin='40000' \
   infile='@asol.list' \
   clobber='no'
 ...
 pset ardlib \
   AXAF_ACIS2_BADPIX_FILE='bpix1[BADPIX2]' \
   AXAF_HRC-I_BADPIX_FILE='bpix1' \
   AXAF_ACIS6_BADPIX_FILE='bpix1[BADPIX6]' \
   AXAF_ACIS3_BADPIX_FILE='bpix1[BADPIX3]' \
   AXAF_ACIS8_BADPIX_FILE='bpix1[BADPIX8]' \
   AXAF_HRC-S_BADPIX_FILE='bpix1' \
   AXAF_ACIS0_BADPIX_FILE='bpix1[BADPIX0]' \
   AXAF_ACIS4_BADPIX_FILE='bpix1[BADPIX4]' \
   AXAF_ACIS1_BADPIX_FILE='bpix1[BADPIX1]' \
   AXAF_ACIS9_BADPIX_FILE='bpix1[BADPIX9]' \
   AXAF_ACIS5_BADPIX_FILE='bpix1[BADPIX5]' \
   AXAF_ACIS7_BADPIX_FILE='bpix1[BADPIX7]'

 mkgrmf \
   mode='hl' \
   wvgrid_arf='compute' \
   order='-1' \
   outfile='heg_-1.rmf' \
   detsubsys='ACIS-S3' \
   obsfile='evt2[EVENTS]' \
   regionfile='pha2[region]' \
   grating_arm='HEG' \
   wvgrid_chan='compute'

 mkgarf \
   mode='hl' \
   verbose='0' \
   asphistfile='s0.asphist[ASPHIST]' \
   order='-1' \
   outfile='s0_heg_-1.arf' \
   sourcepixelx='4039.87' \
   pbkfile='pbk0' \
   detsubsys='ACIS-S0' \
   obsfile='evt2[EVENTS]' \
   sourcepixely='4137.35' \
   grating_arm='HEG' \
   maskfile='none' \
   osipfile='CALDB' \
   engrid='grid(heg_-1.rmf)' \
   dafile='CALDB' \
   clobber='yes'
 ...
 dmarfadd \
   mode='hl' \
   outfile='heg_-1.arf' \
   infile='@heg_-1_garf.list' \
   clobber='yes'
 ...
```

Example:

```
isis> np_mkgarf = make_newpar( ; maskfile="none" );
isis> make_responses( [-1,1], NULL, newpar_mkgarf ) ;
```

See also: `tgextract`, `asphist_multi`, `mkgarf_full`, `ardlib_badpix`, `mkgrmf`

`mkgrmf( [ newpar, ] grating, order, subsys )`

Make a grating response matrix file (RMF) by calling the CIAO tool, `mkgrmf`, for a specified grating, order and detector subsystem on which the zeroth order is imaged.

Default `mkgrmf` parameters can be overridden with the `newpar` container (see `make_newpar` for details).

This assumes that the files `evt2` and `pha2` are in the current directory. The output, *grating_order*`.rmf` is written to the current directory.

grating is one of HEG, MEG, or LEG; only the first character is required, and is case-insensitive.

order is the integer diffraction order; e.g., -1.

`subsys` is the zeroth order detector subsystem (or chip); only `ACIS-S3` or `HRC-S2` are valid values.

Example:

```
isis> mkgrmf( "HEG", 1, "ACIS-S3");

    punlearn mkgrmf

    mkgrmf \
      mode='hl' \
      wvgrid_arf='compute' \
      order='1' \
      outfile='heg_1.rmf' \
      detsubsys='ACIS-S3' \
      obsfile='evt2[EVENTS]' \
      regionfile='pha2[region]' \
      grating_arm='HEG' \
      wvgrid_chan='compute'
```

See also: `make_responses`, `tg_resolve_events`, `tgextract`

`mkgarf_full( [newpar,] order, x, y, detsubsys_list, grating )`

Run a series of `mkgarf` calls over all the detector chips in the array and combine them into a single grating Ancillary Response File (ARF, the effective area).

`order`: diffraction order; an integer;

`x,y`: The zeroth order sky pixel coordinate;

`detsubsys_list`: detector subsystems, or chips; an array of strings; e.g., `["ACIS-S3", "ACIS-S4", "ACIS-S5"]`

`grating`: grating type, `"HEG", "MEG", or "LEG"`

Example (with abbreviated output):

```
isis> mkgarf_full( -1, 4096, 4096,
        ["ACIS-S0", "ACIS-S1", "ACIS-S2", "ACIS-S3"], "HEG");
```

```
punlearn mkgarf
punlearn dmarfadd

mkgarf \
  mode='hl' \
  verbose='0' \
  asphistfile='s0.asphist[ASPHIST]' \
  order='-1' \
  outfile='s0_heg_-1.arf' \
  sourcepixelx='4096' \
  pbkfile='pbk0' \
  detsubsys='ACIS-S0' \
  obsfile='evt2[EVENTS]' \
  sourcepixely='4096' \
  grating_arm='HEG' \
  maskfile='msk1' \
  osipfile='CALDB' \
  engrid='grid(heg_-1.rmf)' \
  dafile='CALDB' \
  clobber='yes'
mkgarf \
  mode='hl' \
  verbose='0' \
  asphistfile='s1.asphist[ASPHIST]' \
  order='-1' \
  outfile='s1_heg_-1.arf' \
  sourcepixelx='4096' \
  pbkfile='pbk0' \
  detsubsys='ACIS-S1' \
  obsfile='evt2[EVENTS]' \
  sourcepixely='4096' \
  grating_arm='HEG' \
  maskfile='msk1' \
  osipfile='CALDB' \
  engrid='grid(heg_-1.rmf)' \
  dafile='CALDB' \
  clobber='yes'
...

echo 's0_heg_-1.arf
s1_heg_-1.arf
s2_heg_-1.arf
s3_heg_-1.arf' > heg_-1_garf.list

dmarfadd \
  mode='hl' \
  outfile='heg_-1.arf' \
  infile='@heg_-1_garf.list' \
  clobber='yes'
```

`mkgarf_multichip( [newpar,] order, x, y, detsubsys_list, grating )`

Run `mkgarf` for a list of chips. This is similar to `mkgarf_full`, but it does not run dmarfadd to combine ARF s.

`mkgarf_chip( [newpar,] order, x, y, detsubsys, grating )`

Run `mkgarf` to make a grating ARF for a single chip specified by `detsubsys`, for the order, source location, and grating as specified by `order`, `x`, `y`, an `grating`, respectively.

Default parameters for `mkgarf` can be overridden with the `newpar` container. See `make_newpar` for details.

Assumes that aspect histogram, parameter block, and grating response matrix files exist in the current directory with proper names. See the example.

Example:

```
isis> mkgarf_chip( -1, 4096.5, 4100.2, "ACIS-S1", "MEG");

punlearn mkgarf

mkgarf \
  mode='hl' \
  verbose='0' \
  asphistfile='s1.asphist[ASPHIST]' \
  order='-1' \
  outfile='s1_meg_-1.arf' \
  sourcepixelx='4096.5' \
  pbkfile='pbk0' \
  detsubsys='ACIS-S1' \
  obsfile='evt2[EVENTS]' \
  sourcepixely='4100.2' \
  grating_arm='MEG' \
  maskfile='msk1' \
  osipfile='CALDB' \
  engrid='grid(meg_-1.rmf)' \
  dafile='CALDB' \
  clobber='yes'
```

See also: `make_responses`, `mkgarf_full`, `mkgarf_cleanup`, `mkgarf_multichip`

`dmarfadd( [newpar,] order, grating )`

Run `dmarfadd` using the order and grating to construct the list of filename for the ARF s for each chip, grating, and order.

`mkgarf_cleanup( order, detsubsys_list, grating )`

Removes the temporary ARF files. This should only be run after `dmarfadd`.

`ardlib_badpix( [newpar] )`

Set the `ardlib.par` bad-pixel parameters.

See also: `make_responses`, `mkgarf`

`asphist( [newpar,] chip, det )`

Make an aspect histogram for a specified chip and detector.

See also: `make_responses`, `mkgarf_chip`, `aphist_multi`

`asphist_multi( [newpar,] chips, det )`

Make a set of aspect histograms for a list of chips and a given detector.

See also: `make_responses`, `mkgarf_chip`, `aphist`

11.5 Lightcurves

`lightcurve( [newpar,] [tbin_ks] )`

Generate counts and count-rate light curves from diffracted events. Use the stored configuration to determine the instrument-dependent parameters. Only first orders are used, if ACIS.

For ACIS, the `aglc` (ACIS Grating Light Curve) ISIS package is used, since it properly handles frame drops and exposure, which is important for CC-mode data. For HRC, the CIAO tool, `dmextract`, is adequate.

The default time bin is 1 ks, which can be changed with the optional parameter, `tbin_ks`.

Parameters can be overlayed for HRC-only with the `newpar` argument.

The output file is called `lc` and is a "standard" FITS light curve table (though the HRC file and ACIS files have slightly different columns).

If the detector is HRC-S, a background light curve is also extracted and written to `lc_bg`.

The input file, `evt2`, must be in the current directory.

This calls one of `lightcurve_ha`, `lightcurve_la`, or `lightcurve_lh` and `lightcurve_bg_lh`.

`lightcurve_acis( [tbin_ks] )`

Generate counts and count-rate light curves for an ACIS observation, using either LETG or HETG, as determined from the stored instrumental configuration. The default bin size is 1 ks.

This calls either `lightcurve_ha` or `lightcurve_la`.

`lightcurve_ha( tbin_ks )`

Generate a light curve for HETG/ACIS-S, using `aglc`.

`lightcurve_la( tbin_ks )`

Generate a light curve for LETG/ACIS-S, using `aglc`.

`lightcurve_lh( [newpar],[tbin_ks] )`

Generate a light curve for LETG/HRC-S, using `dmextract`.

`lightcurve_bg_lh( [newpar],[tbin_ks] )`

Generate a background light curve for LETG/HRC-S, using `dmextract`.

Part III
𝒯𝒢Cat Detailed Examples

. . . coming later . . .

12 HETG/ACIS, TE, narrow regions

13 HETG/ACIS, Level 1a only

14 ACIS, CC, custom osip

15 ACIS/HETG, summary plots

16 LETG/HRC-S, responses only

17 HETG/ACIS-S, higher order responses

18 HETG/ACIS-S, custom QE for calibration

19 HETG/ACIS-S, save and re-run