GHOST Users Manual

Release 0.9.0

Marc White, Kathleen Labrie & the GHOST DRS Team

May 19, 2017

Contents

CHAPTER 1

Overview of GHOST

Note: This section of documentation is a summary/transcription of the GHOST Concept of Operations Document (ConOps) as at November 2015. It should be reviewed and updated once the instrument is actually built and commissioned.

Description of the Instrument

GHOST is a fibre-fed echelle spectrograph. It has a wide variety of observing modes tuned to a range of science cases.

GHOST comprises two positioner arms, "IFU 1" and "IFU 2". The layout of each focal plane is shown below. Each comprises multiple micro-lens arrays (black hexagons) feeding fiber bundles for targeting science fields at standard (large hexagons in IFU 1 and 2) or high (small hexagons in IFU 1) spectral resolution, peripheral guide fibers surrounding the science fields (red hexagons), and dedicated sky fibers at a fixed offset position a few arcseconds from the science fields (cyan hexagons). Each positioner can access different halves of the 7.5 arcminute field (as shown in Figure 4), with a 16" common region of overlap.

Warning: Find image

The instrument has two distinct observing modes. In 'standard' mode, up to two targets can be observed simultaneously using the large fibre bundles on IFU1 and IFU2. In 'high resolution' mode, a single target can be observed using the high-resolution bundle on IFU1. A ThXe calibration lamp can also be supplied simultaneously with the observations. Sky fibre bundles are provided for 'standard' mode on IFU 2, and for 'high-resolution' mode on IFU 1.

Mode	Standard Resolution	High Resolution
Spectral coverage	363-950nm, simultaneous	363-950nm, simultaneous
Spectral resolution	50,000	75,000
Radial velocity precision	600 m/s	10 m/s
Multiplexing	Dual targets/beam switching (82" minimum separation)	Single target
Patrol field	7.5' semicircle (16" overlap between IFUs)	7.5'
IFU plane size	1.2"	1.2"
IFU element number and size	7×0.4"	19×0.2"
Sky fibres	3×0.4" (on IFU 1)	7×0.2" (on IFU 2)
Approx. limiting Vega magnitude[1]	17.4 - 17.8	17.0 - 17.4

The GHOST control software positions the IFUs as required during observations. A mask in the slit injection unit blocks light from the unused fibre bundles.

Instrument Operating Modes

The following table is an overview of the instrument observing modes available to users, which are selecting via the Gemini Observing Tool (OT).

Configuration	Focal Plane	Resolution	Detector binning[2]	Guiding[3]	Approx. B mag.
Two-target	IFU1 & IFU2	50k	1x2	P2 & guide fibres	< 18
Two-target (faint)	IFU1 & IFU2	50k	1x4	P2	≥ 18
Two-target (very faint)	IFU1 & IFU2	43k	2x4	P2	≥ 18
Beam-switching	IFU1 & IFU2	50k	1x2	P2 & guide fibres	< 18
Beam-switching (faint)	IFU1 & IFU2	50k	1x8	P2	≥ 18
Beam-switching (v. faint)	IFU1 & IFU2	43k	2x8	P2	≥ 18
High-resolution	IFU1	75k	1x1	P2 & guide fibres	< 17
High-resolution (faint)	IFU1	75k	1x8	P2 & guide fibres	$17 - 18$
High-resolution (PRV)	IFU1 & ThXe	75k	1x1	P2 & guide fibres	< 16

Two-target modes

In standard spectral resolution mode, two targets can be observed simultaneously using the standard-resolution fibre bundles on IFU1 and IFU2, and the associated sky bundle on IFU1. Targets are specified to the OT using absolute astronomical coordinates; a guide star is also needed for the peripheral wave-front sensor (PWFS) guiding. Care needs to be taken to configure the instrument such that the PWFS does not vignette a science IFU.

For standard two-target modes, targets are expected to be bright enough that the PSF edges can be used for guiding via the guide fibres attached to the science bundles. For faint/very faint mode, guiding is by PWFS only. Guide fibres

[1] Can achieve S/N ratio of 30 in 1 hour at 450 nm.
[2] Reported as (binning in the spectral direction) x (binning in the slit direction)
[3] P2 corresponds to peripheral wave-front sensor (PWFS) guiding.

can also be disabled in standard two-target mode if crowded fields cause the guiding to be inaccurate.

In faint/very-faint mode, a larger detector binning is used to reduce the impact of read noise.

Beam-switching modes

In regions of low target density (i.e. where there is a single target within the GHOST field-of-view), the two standard-resolution IFUs may be beam-switched to provide continuous target observation, whilst alternating each IFU between the target and an offset sky position. This facilitates accurate sky subtraction by differencing sequential frames, avoides the resampling of bright sky lines or detector artefacts, and elimiates the effects of potential flat fielding errors and differential fibre throughputs. This is particularly useful for faint targets and the 'red' camera, where there are numerous time-variable sky lines.

The Gemini OT will automatically set diametrically opposed offset conditions for sky measurements, to allow beam-switching to be accomplished using telescope motion alone. However, in the case that this is inappropriate (e.g. crowded fields, or where the PWFS may vignette a science detector using the default configuration), it is possible to explicitly specify sky positions. This inflicts a time penalty, as the IFU positions will need to be reconfigured.

In the faint and very-faint modes, larger detector binning is used, and guiding via the GHOST fibres is disabled. Guide fibres may be used in the standard beam-switching mode, although like the two-target mode, this can be disabled if necessary.

High-resolution modes

High-resolution modes use the high-resolution science fibre bundle on IFU1. A high-resolution sky fibre bundle is on IFU2, and can be positioned independently of IFU1 for simultaneous sky observations. The use of a single science field provides maximum flexibility for the positioning of IFUs so as to avoid vignetting by the PWFS, and maximizes the patrol radius for selecting PWFS guide stars. Spectral binning in the spectral direction is not used in this mode, to fully sample the spectral PSF. A factor 2 binning along the slit is optimal.

Warning: This factor 2 binning isn't reflected in the table!

The high-resolution science fibre bundle has six peripheral guide bundles, for guiding using the extended PSF of bright targets. This can be disabled as required, and is disabled by default in faint mode. Eight-pixel binning in the slit direction is also used in faint mode.

For targets requiring the best possible wavelength calibration, a precision radial velocity (PRV) mode is provided. A fibre agitator is used to reduce modal noise introduced to the fibres by stress, strain or imperfections. A ThXe calibration source is may also be fed into an additional high-resolution fibre which is passed to the spectrograph for calibration simultaneous to observations. This source is cycled on and off with a given duty cycle, giving total counts within a given exposure time to be similar in magnitude to the science fibres (and avoiding saturation).

Description of the Data

Note: Will actually need some, you know, data to do this completely.

BPM Flag Encoding

The bad pixel mask (BPM) flag encoding used for GHOST is derived from that used for GHOS, and is summarized below:

Bit	Pixel value	Meaning
0	0	No conditions apply (i.e. good data)
1	1	Generic bad pixel (e.g. region occulted/not illuminated; hot pixel; bad column)
2	2	Highly non-linear pixel response
3	4	Saturated pixel
4	8	Cosmic ray hit
5	16	Invalid data (e.g. all data rejected during stacking)
6	32	Not used.
7	64	Not used.
8	128	Not used.
9	256	SCI pixel value has been replaced via interpolation
10	512	SCI pixel value has been replaced, but **not** via interpolation

CHAPTER 2

Polynomial Model Method

Description of the Spectrograph Model principle

The principle employed in the development of this pipeline relies heavily on a polynomial principle for all the modelling of the spectrograph's characteristics from the data. These include the location of the orders, the wavelength scale and the reciprocal model that converts the sampled slit from the slit viewer to its image on the spectrograph CCD.

The idea is that instead of a traditional empirical extraction where the orders are "scanned" for their location and arc lines are detected blindly, we form a model of the spectrograph and fit the parameters using the flat fields and arcs to measure small changes in the spectrograph on a nightly basis. Then, the extraction process becomes relatively trivial with knowledge of where all the flux is and the wavelength scale is uniquely determined. A simple advantage of this method is that the entire spectrograph is modelled as one, instead of each individual order as a separate entity. Ultimately, measurements such as radial velocity shifts can be determined using a single varying parameter, as opposed to a combination of measured shifts in all orders.

The principle implemented is that of a sum of polynomials of polynomials. This differs from an approach where each physical parameter of the spectrograph is modelled individually and focusses on a series of coefficients that represent various aspects of the CCD images. We thereby minimise the number of required parameters that describe the data.

Mathematical principle

The `polyfit` method uses files containing polynomial coefficients where each line is the coefficients for the polynomials as a function of order, which are then combined as a function of y position on the CCD chip, defined as the CCD pixel numbers in the spectral direction.

For mathematical convininence and correspondence with a testable reference, the polynomials are evaluated with respect to a reference order $m_{\rm ref}$, defaulting as whatever order number is in the middle of the range used for each arm, and as a function of the middle pixel on the chip y_{middle}.

The functional form is:

$$F(p) = p_0(m) + p_1(m) * y' + p_2(m) * y'^2 + ...$$

with $y' = y - y_{middle}$, and:

$$p_0(m) = q_{00} + q_{01} * m' + q_{02} * m'^2 + ...$$

with $m' = m_{\rm ref}/m - 1$

In this functional form, $F(p)$ is whatever aspect we wish to model. In the specific example of GHOST, it will be the x position (defined in the spatial direction) in the first instance, but this same method is then used for the wavelength scale, and all three aspects of the slit image on the chip (spatial direction magnification scale, spectral direction magnification scale and rotation), all of which are expected to change as a function of order and position along the order.

This means that the simplest wavelength scale spectrograph model should have:

- q_{00} : central wavelength of order m_ref
- q_{01} : central wavelength of order m_ref
- q_{10} : central_wavelength/R_pix, with R_pix the resolving power / pixel.
- q_{11} : central_wavelength/R_pix, with R_pix the resolving power / pixel.

... with everything else approximately zero.

Please note that the order of polynomials is left undefined. The code that handles these parameters is identical and left generalised since each aspect (x position, wavelength, etc) may require a different number of variables to fully describe the problem.

Description of model file contents

In the case of the x position, using default file *xmod.fits*, the contents of this file are as follows:

$$X_{mod} = \begin{bmatrix} q_{24} & q_{23} & q_{22} & q_{21} & q_{20} \\ q_{14} & q_{13} & q_{12} & q_{11} & q_{10} \\ q_{04} & q_{03} & q_{02} & q_{01} & q_{00} \end{bmatrix} \tag{2.1}$$

The non standard way to define the variables within the files and imported array is related to the way numpy's poly1d function takes inputs, with the highest order coefficient first.

In the case of x position, the coefficients represent:

- q_{00} : x position of the middle of the reference order.
- q_{01} : linear term coefficient for order spacing
- q_{02} : quadratic term coefficient for order spacing
- q_{10} : common rotation term for all orders
- q_{11} : linear term coefficient for order rotation
- q_{12} : quadratic term coefficient for order rotation
- q_{20} : common curvature term for all orders
- q_{21} : linear term coefficient for order curvature
- q_{22} : quadratic term coefficient for order curvature

... with everything else approximately zero.

CHAPTER 3

Recipes for GHOST

Typical Processing Flows

Here we review some of the typical processing workflows for GHOST data reduction. For this discussion, it is assumed you've already installed the latest Ureka package and made a local clone of the `ghostdr` Hg repository. (For illustrative purposes, the text below assumes the clone's working copy root is `wc/`.)

Furthermore, for the commands given below to work properly, you must:

1. initialize the Ureka environment: `ur_setup`
2. create a symlink named `wc/externals/gemini_python/astrodata_GHOST` pointing to `wc/astrodata_GHOST`,
3. add `wc/externals/gemini_python` to the beginning of your `PYTHONPATH`, and
4. add `wc/externals/gemini_python/astrodata/scripts` and `wc/externals/gemini_python/recipe_system/apps` to the beginning of your `PATH`

Generating a Bias Calibration frame

To generate a bias calibration frame you need 2 or more GHOST bias frames from the same arm. Until the instrument is live, you can use the GHOST simulator to generate this data. Its `testsim.py` script will create several types of frames, including 3 bias frames for each arm, 3 darks for each arm, and 3 flats for each arm and resolution combination. You can comment out the generation of non-bias frame types to speed things up.

Once you have a few biases of the same arm to work with, generate a file list using the `typewalk` utility. The following command assumes you have generated several red arm biases (if you don't specify either `GHOST_RED` or `GHOST_BLUE`, you may get mixed red and blue frames which don't stack well!):

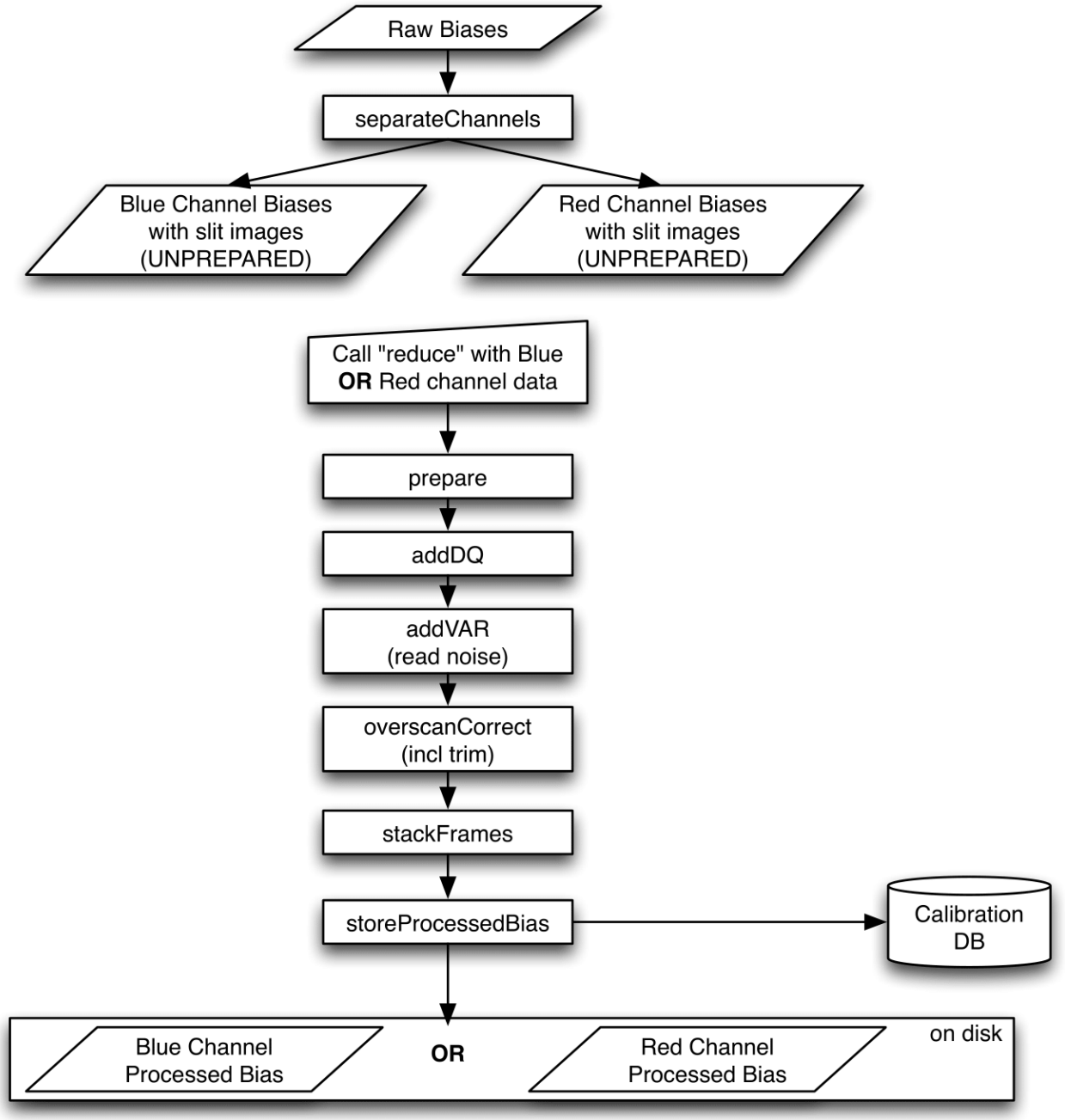
```
typewalk --types GHOST_BIAS GHOST_RED --dir <path_to>/data_folder -o bias.list
```

Now you are ready to generate a bias calibration frame. The following command (which runs the `makeProcessedBiasG` Gemini recipe behind the scenes) will stack the bias frames in listed `bias.list` and store the finished bias calibration in `calibrations/storedcals/`:

```
reduce @<path_to>/bias.list
```

Don't forget the @ character in this line, e.g. if <path_to> is `data` then this command should be `reduce @data/bias.list`. The @ parameter is a legacy from IRAF, and tells `reduce` that you're passing a list of filenames instead of a data file. This code call will place a file named `bias_1_red_bias.fits` in the `calibrations/storedcals` directory of your present working directory.

The whole process behind Gemini's `makeProcessedBias` recipe is documented in the following flowchart (thanks Kathleen Labrie):



storeProcessedBias will only work with internal calibration database for now.
We can look into making it work with the local/user calibration database. TBD.

Generating a Dark Calibration Frame

The procedure for generating a dark calibration frame is broadly similar to making a bias calibration frame. However, the type to be passed to `typewalk` should be `GHOST_DARK` instead of `GHOST_BIAS` (in addition to the necessary

GHOST_RED/GHOST_BLUE type):

```
typewalk --types GHOST_DARK GHOST_RED --dir <path_to>/data_folder -o dark.list
```

Assuming typewalk has output your list of dark frames to dark.list, attempting to run:

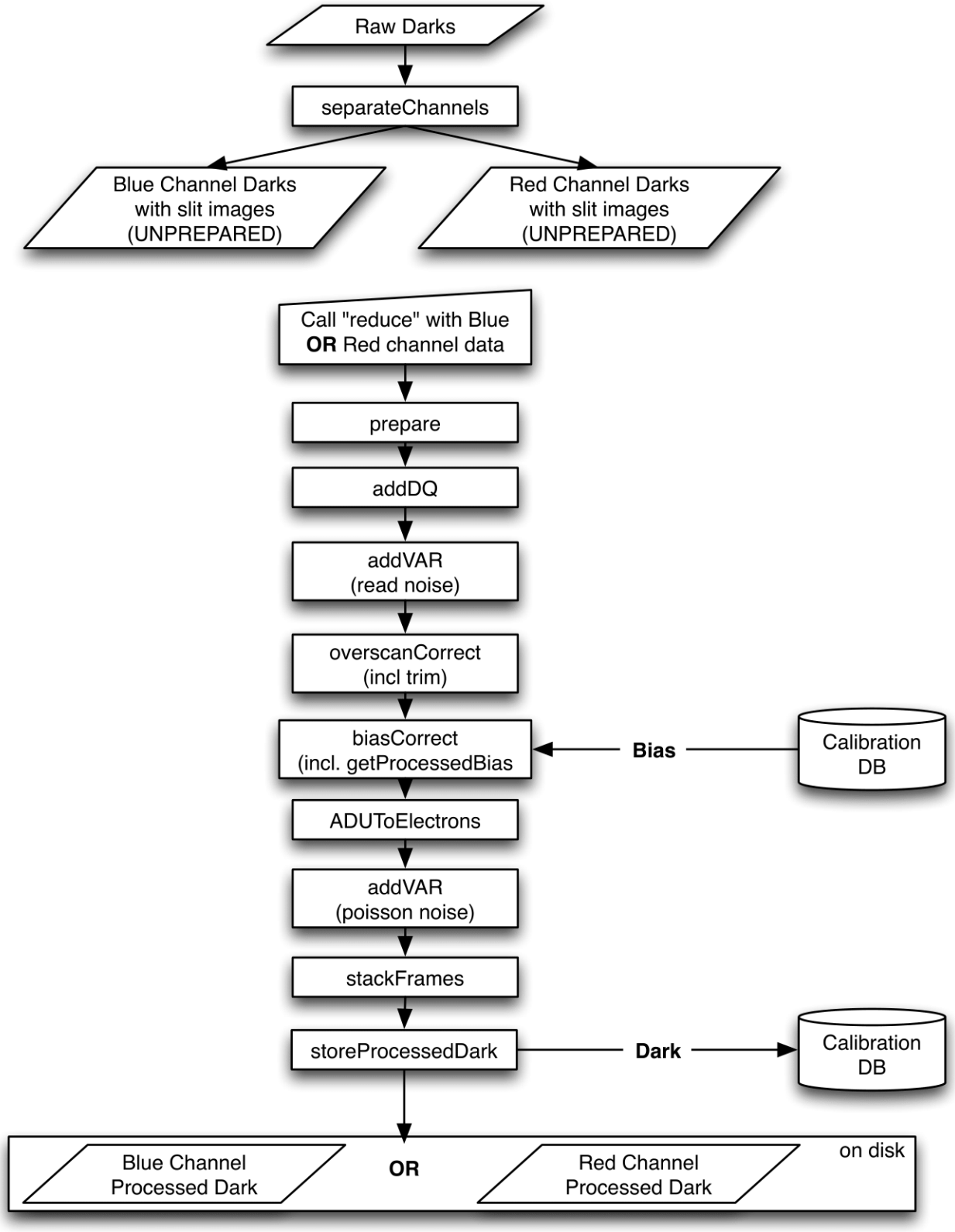
```
reduce @<path_to>/dark.list
```

will fail. This is because the framework cannot currently find calibrations stored on disk (it uses a much more complicated lookup scheme). The workaround for the time being is to force it to look on disk in a particular area using the --override_cal option:

```
reduce @<path_to>/dark.list --override_cal processed_bias:calibrations/storedcals/
↪bias_1_red_bias.fits
```

(Depending on your specific bias.list contents, your bias calibration under your calibrations/storedcals directory may have a different name, so double- check.) This command will place a file dark95_1_red_dark.fits into the calibrations/storedcals directory.

The whole process behind Gemini's makeProcessedDark recipe is documented in the following flowchart (thanks Kathleen Labrie):



storeProcessedDark will only work with internal calibration database for now.
We can look into making it work with the local/user calibration database. TBD.

Generating a Flat Calibration Frame

The procedure for generating a flat field calibration frame is similar to creating a dark or bias, although you have to `typewalk` over GHOST_FLAT files instead, e.g.:

```
typewalk --types GHOST_FLAT GHOST_RED GHOST_HIGH --dir <path_to>/data_folder -o flat.
↪list
```

(Note this is the first place where we have to explicitly specify the resolution mode/type of the object file we ultimately intend to reduce.) Then, when you call `reduce` on the `flat.list`, you must provide both the bias and dark file path explicitly:

```
reduce @<path_to>/flat.list --override_cal processed_bias:calibrations/storedcals/
↪bias_1_red_bias.fits processed_dark:calibrations/storedcals/dark95_1_red_dark.fits
```

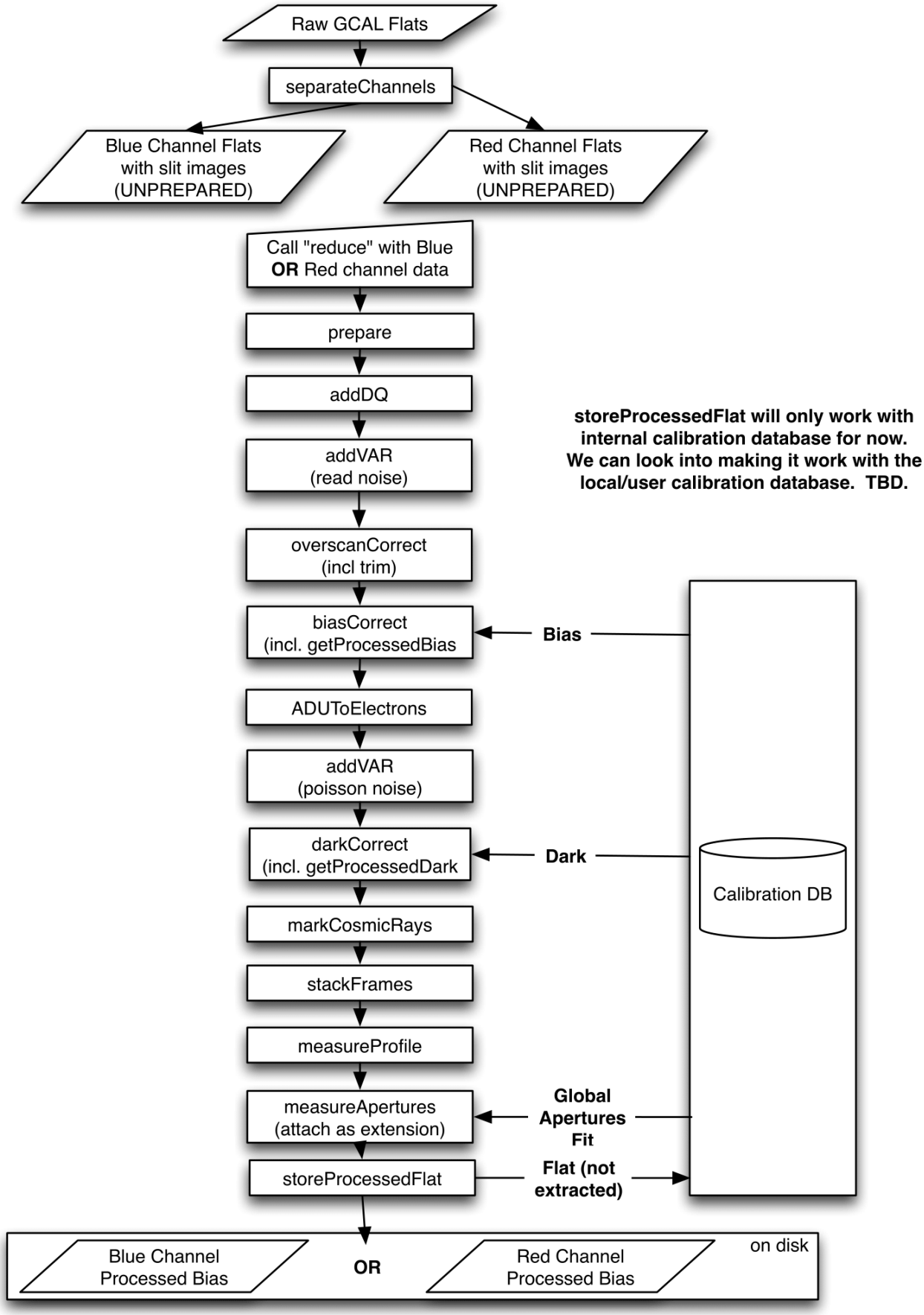
(or whatever the filename of the processed dark turns out to be).

After the flat field has been created, the spectrograph apertures are fit using a `polyfit` approach. The RecipeSystem will read in the appropriate aperture model from the `lookups` system, fit it to the flat field, and store the resulting model in the calibrations system.

The selection of the appropriate `polyfit` model to start with is determined by the spectrograph arm, resolution, and the date the observations are made on. Ideally, there will only be one model per arm and resolution combination; however, spectrograph maintenance (i.e. dis- and re-assembly) may result in the model changing at a specific point in time. Therefore, the RecipeSystem *should* (see below) automatically choose the most recent applicable model for the dataset being considered.

Note: Date-based model selection is currently not implemented - instead, only a single model is provided for each arm/resolution combination. This is sufficient for testing involving the simulator data. Date-based selection will be implemented soon.

The process behind `makeProcessedFlatG` is summarized in the following flowchart (thanks Kathleen Labrie):


Raw GCAL Flats
separateChannels
Blue Channel Flats with slit images (UNPREPARED)
Red Channel Flats with slit images (UNPREPARED)
Call "reduce" with Blue OR Red channel data
prepare
addDQ
addVAR (read noise)
overscanCorrect (incl trim)
biasCorrect (incl. getProcessedBias
ADUToElectrons
addVAR (poisson noise)
darkCorrect (incl. getProcessedDark
markCosmicRays
stackFrames
measureProfile
measureApertures (attach as extension)
storeProcessedFlat
storeProcessedFlat will only work with internal calibration database for now. We can look into making it work with the local/user calibration database. TBD.
Bias
Dark
Global Apertures Fit
Flat (not extracted)
Calibration DB
Blue Channel Processed Bias
OR
Red Channel Processed Bias
on disk

Note: This is the originally-envisaged implementation of `makeProcessedFlatG`. It has since been decided that Gemini will guarantee that Gemini Observatory will always take at least three flat fields per arm per observation, which means that `rejectCosmicRays` is not required; `stackFrames` will remove almost all cosmic rays.

Generating an Arc Calibration Frame

Warning: You *must* have performed a full slit viewer reduction before attempting to make an arc calibrator - the results of the slit flat and slit image reduction are required to make the profile extraction and subsequent wavelength fitting work. See *Reducing Slit Viewing Images* for details.

Making an arc calibration frame is similar to the previous calibration steps. The correct type to `typewalk` across is `GHOST_ARC`:

```
typewalk --types GHOST_ARC GHOST_RED GHOST_HIGH --dir <path_to>/data_folder -o arc.
↪list
```

Additional calibrators required are reduced slit viewer flats and slit viewer images, as well as the aperture fit made during the generation of the flat calibration image:

```
reduce @<path_to>/arc.list --override_cal processed_bias:calibrations/storedcals/bias_
↪1_red_bias.fits processed_dark:calibrations/storedcals/dark95_1_red_dark.fits␣
↪processed_slit:calibrations/storedcals/obj95_1.0_high_SLIT_stack_slit.fits␣
↪processed_slitflat:calibrations/storedcals/flat95_high_1_SLIT_stack_slitFlat.fits␣
↪processed_xmod:calibrations/storedcals/GHOST_1_1_red_high_xmodPolyfit.fits
```

Arc reduction not only generates a reduced arc image and places it in the calibrations directory, but also uses the `polyfit` module to extract the flux profiles of the object/sky fibres in the input image. It then uses this fit, and a line set stored in the RecipeSystem lookups system, to make a wavelength fit to the arc image. This fit is also stored in the calibrations directory/system.

Reducing an Object frame (Spectra)

The GHOST simulator produces object spectra frames like `obj95_1.0_std_red.fits` whose names follow this convention: `obj{exptime}_{seeing}_{resolution}_{arm}.fits`. If you run `typewalk` on the folder containing these, you'll see that they are identified as `GHOST_OBJECT`:

```
typewalk --dir <path_to>/data_folder
```

This informs the reduction framework to run the `reduceG` GHOST recipe on them. which should run to at least the `flatCorrect` step now that you have dark and bias calibration frames (for the moment, we have commented the remaining steps out of the `reduceG` recipe so it will complete successfully):

```
reduce <path_to>/data_folder/obj95_1.0_high_red.fits
```

The above command will fail due to the faulty calibrations lookup. Again, we need to use the `--override_cal` option:

```
reduce <path_to>/data_folder/obj95_1.0_high_red.fits --override_cal processed_
↪bias:calibrations/storedcals/bias_1_red_bias.fits processed_dark:calibrations/
↪storedcals/dark95_1_red_dark.fits processed_flat:calibrations/storedcals/flat95_
↪high_1_red_flat.fits
```

This produces a `obj95_1.0_high_red_flatCorrected.fits` (or similar) file, a bias, dark and flat corrected GHOST spectrum frame.

Warning: The primitive `rejectCosmicRays` would normally be called as part of `reduceG`, after the `darkCorrect` step. It is currently commented out - the underlying LACosmic algorithm is working, but aperture removal/re-instatement is required to avoid accidentally flagging spectral peaks and the edges of orders as cosmic rays, and this has yet to be implemented.

Reducing Slit Viewing Images

Reducing slit viewer images is very similar to reducing standard images, including steps to generate bias, dark and flat calibration frames, plus a final step to process the slit viewer frames (which removes cosmic rays and computes the mean exposure epoch). The first step, computing the bias calibrator, may be skipped in favour of simply pointing to a slit bias frame (of type `GHOST_SLITV_BIAS`). Or, follow these steps to produce one by stacking multiple frames together:

```
typewalk --types GHOST_SLITV_BIAS --dir <path_to>/data_folder -o slit_bias.list
reduce @<path_to>/slit_bias.list
```

The next step is to generate the dark calibrator. Follow these steps to produce one:

```
typewalk --types GHOST_SLITV_DARK --dir <path_to>/data_folder -o slit_dark.list
reduce @<path_to>/slit_dark.list --override_cal processed_bias:calibrations/
↪storedcals/bias_1_SLIT_stack_slitBias.fits
```

Now generate the flat calibrator. For this you will now need to specify an additional type to `typewalk` that identifies the resolution of the data that you wish to process (as mixing resolutions would be nonsensical). Follow these steps as an example:

```
typewalk --types GHOST_SLITV_FLAT GHOST_HIGH --dir <path_to>/data_folder -o slit_flat_
↪high.list
reduce @<path_to>/slit_flat_high.list --override_cal processed_bias:calibrations/
↪storedcals/bias_1_SLIT_stack_slitBias.fits processed_dark:calibrations/storedcals/
↪dark95_1_SLIT_stack_slitDark.fits
```

The final step is to use all of the above calibrators in a call to `reduce` a set of slit viewer images taken concurrently with a science frame, usually found in files named like `obj95_1.0_high_SLIT.fits` (following this convention: `obj{exptime}_{seeing}_{resolution}_SLIT.fits`). If you run `typewalk` on the folder containing these, you'll see that they are identified as `GHOST_SLITV_IMAGE`. This informs the reduction framework to run the `makeProcessedSlitG` GHOST recipe on them. Run the reduction as follows (note that the flat is provided to `--override_cal` as `process_slitflat` and not simply `processed_flat`):

```
reduce <path_to>/data_folder/obj95_1.0_high_SLIT.fits --override_cal processed_
↪bias:calibrations/storedcals/bias_1_SLIT_stack_slitBias.fits processed_
↪dark:calibrations/storedcals/dark95_1_SLIT_stack_slitDark.fits processed_
↪slitflat:calibrations/storedcals/flat95_high_1_SLIT_stack_slitFlat.fits
```

Other Processing Flows

include scientific flow charts, include associated recipes

CHAPTER 4

Tips and Tricks for Processing GHOST

Some title depending on content

Describe the quirks of GHOST data, give tips and tricks, what to watch for. Screenshots are encourages.

Example

Some other topic

Example

CHAPTER 5

Issues and Limitations

CHAPTER 6

Primitives for GHOST

correctSlitCosmics

Purpose

This primitive replaces CR-affected pixels in each individual slit viewer image (taken from the current stream) with their equivalents from the median frame of those images.

Inputs and Outputs

`correctSlitCosmics` takes no particular configuration inputs.

Algorithm

The incoming stream of AstroData objects, representing bias-/dark-corrected slit viewer frames, is first used to produce a per-pixel median absolute deviation (MAD) frame for clipping cosmic ray (CR) outliers. A median frame is then separately produced (from the inputs) and subtracted from each in turn. CR-impacted pixels are those where the difference residual exceeds 20x the MAD, which are then replaced with their equivalents from the median slit viewer frame.

Issues and Limitations

For short duration science exposures there may be too few coincident slit frames to produce a meaningful MAD frame for CR detection.

extractProfile

Purpose

TODO

Inputs and Outputs

TODO

Algorithm

TODO

Issues and Limitations

TODO

findApertures

Purpose

This primitive will take an existing `polyfit` model for the aperture locations in a frame, and fit the model to the observed data.

Note that this primitive is only run on prepared flat fields (i.e. images that have been run through the `makeProcessedFlatG` recipe). The primitive will abort if passed any other type of file.

Inputs and Outputs

`findApertures` takes no particular configuration inputs.

Algorithm

The RecipeSystem will automatically determine the appropriate initial polyfit model, based on the arm, resolution and date of the observation.

Note: Date selection of polyfit models has yet to be implemented.

The primitive will instantiate an `Arm` object from the `polyfit.ghost` module, which is included as a part of `astrodata_GHOST`. This `Arm` contains all the functions for fitting the aperture positions of the data.

The `Arm` object then makes the following three steps:

- An initial model of the spectrograph is constructed based on the parameters read-on from the lookup system;
- The reduced flat-field is convolved with the slit profile of the instrument, determined from the slit viewing camera

- The initial model is fitted to the result of the convolved flat field, which represents the location of the middle of each order. The result of this fit is then used in the spectra extraction as the basis for the location of the orders.

A `polyfit` model FITS file is then written out to the calibration system.

Issues and Limitations

There is a placeholder retrieval function for getting back a fitted `polyfit` model, but there are no primitives yet for applying this model to data.

Future versions of this fitting procedure will include an extra polynomial describing the change in the spatial scale of the fiber images as a function of order location on the CCD. The current lack of such feature currently results in a slight innaccuracy of the fit on the edge of orders which are close together. This is due to the fact that the convolution with a fixed profile results is slight overlap at those locations.

fitWavelength

Purpose

TODO

Inputs and Outputs

TODO

Algorithm

TODO

Issues and Limitations

TODO

fork

Purpose

This primitive creates a new stream by copying the current stream's inputs to the outputs of the new stream. Has the same effect as (but without the disk write penalty incurred by) the following construct:

```
addToList(purpose=save_to_disk)
getList(purpose=save_to_disk, to_stream=new_stream_name)
```

Inputs and Outputs

The only parameter `fork` takes is `newStream`, the name of the new stream to be formed.

Issues and Limitations

Be careful not to specify the parameter `newStream` as simply `stream` as this is a reserved parameter name with special meaning to the recipe framework and odd behaviour results when used.

mosaicADdetectors

Purpose

This primitive mosaics the SCI frames of the input images, along with the VAR and DQ frames if they exist.

Inputs and Outputs

`tile`: bool (default: False), tile images instead of mosaic (mosaic includes transformations)

`dq_planes`: bool (default: False), transform the DQ image, bit plane by bit plane

Algorithm

This primitive overrides the one of the same name inherited from `GEMINIPrimitives`. The override logic is identical to the original except for two hacks introduced just prior to calling the mosaicing routine, `gemini_mosaic_function`, and undone immediately after. These are necessary in order to trick the underlying mosaicing machinery into working for GHOST data, and are as follows:

- First, we add 'GSAOI' to the internal array of types for our input AstroData object (which contains the extensions we want to mosaic). This bypasses the GSAOI-/GMOS-only check built into `gemini_mosaic_function`. Afterwards we reset the type list back to normal by invoking `AstroData.refresh_types` on the resulting, mosaiced, AstroData object.
- Secondly, we set the `INSTRUME` PHU keyword to `../../../astrodata_GHOST/ADCONFIG_GHOST/lookups`. This tricks `set_geo_values` (called by `gemini_mosaic_function`) into referencing the `geometry_conf.py` file under our own `astrodata_GHOST/lookups` folder. As soon as the mosaicing routine returns we reset `INSTRUME` back to 'GHOST' (and in fact this must be done before the call to `refresh_types` mentioned above).

Until such time as `astrodata_GHOST` has been merged into `astrodata_Gemini` (like GMOS and GSAOI have been), or `astrodata_Gemini` is refactored, the above two hacks will be necessary in order for GHOST to make use of Gemini's mosaicing mechanism.

Issues and Limitations

Requires ugly hacks until integration with, or refactoring of, `astrodata_Gemini` is completed.

Primitive #1 (alphabetical)

Purpose

Inputs and Outputs

Algorithm

Issues and Limitations

Primitive #2 (alphabetical)

Purpose

Inputs and Outputs

Algorithm

Issues and Limitations

processSlits

Purpose

For each CR-corrected slit viewer frame taken from the current stream, this primitive extracts the object profiles within and uses them to flux-weight the image's offset (relative to the coincident science frame) and then uses that to compute the mean exposure epoch for the entire series of inputs.

Inputs and Outputs

`flat`: string or None (default: None) - The name of the processed slit viewer flat field image. If not provided, or if set to None, the flat will be taken from the `processed_slitflat` override_cal command line argument.

Algorithm

For each individual CR-corrected frame taken from the input, a `SlitViewer` object is instantiated for each spectrograph arm using the frame and the passed- in flat field image. The `SlitViewer` objects then each provide, via the `object_slit_profiles()` method, a 2 element array where each element is a 1-dimensional array representing a single object profile, un-normalised, sky-corrected, and summed across (not along) the slit.

In the case of high resolution data, the Th/Ar profiles are discarded, after which the flux for the remaining object profiles, for both arms, is summed together. This individual frame sum is then scaled by its percentage overlap with the science frame (computed from UTC start/stop times for both the individual frame and the science frame), which in turn is used to weight the frame's offset (in seconds) from the start of the science exposure. [Note that only the science frame's UTC start/stop times are present for this process, not its frame data.]

Finally the sum of the weighted offsets is divided by the sum of the weights to provide the mean offset from the start of the science exposure. This is added to the science exposure UTC start time to produce the mean exposure epoch, which in turn is written into the header of each CR-corrected output frame, in the `AVGEPOCH` keyword.

Issues and Limitations

The UTC start/stop times of the coincident science frame are currently obtained from the UTSTART/UTEND PHU keywords, having been written there by our splitter primitive (or the simulator with the -split option used). In the context of a standalone slit viewer frame, this may be considered improper usage of these keywords.

promote

Purpose

This primitive is responsible for taking a slit viewer MEF (that results as the output from the observation bundle splitter primitive) and "promoting" all the slit viewer extensions therein to full AstroData objects with various manipulations so that they will pass various checks and otherwise work nicely with the rest of the recipe.

Note that this primitive is only meant to be run on raw (unprocessed) slit viewer MEFs produced by the splitter primitive (or by the simulator with the -split option supplied).

Inputs and Outputs

`promote` takes no particular configuration inputs.

Algorithm

The `promote` primitive performs these manipulations on the slit viewer extensions:

- It uses AstroData slicing (also known as "sub-data") to initially produce a separate AstroData object for each extension, then also makes a copy of the PHU for each resultant AstroData object (since sub-data references a single copy of the original PHU by default). In this manner, fully independent AstroData objects are created representing each extension. This is done to bypass the extension count check we inherit from `GMOSPrimitives.validateData`, which is suitable for processing GHOST's red and blue arm data, but not for the slit viewer MEFs which may contain any number of extensions.
- The above manipulation causes the resulting AstroData objects to all have the same ORIGNAME in their PHUs. This becomes an issue when we leverage the existing `StackPrimitives.stackFrames` primitive, as it must first (being IRAF-based) write its inputs to disk. As it uses ORIGNAME to do so, clashes would result, so we make ORIGNAME unique by simply appending an incrementing number.
- Finally, we also reset the EXTVER of all extensions in the AstroData objects resulting from the first manipulation back to 1 (since now there is only a single extension in each AstroData object).

Issues and Limitations

None.

rejectCosmicRays

Purpose

This primitive will mask cosmic rays in a GHOST data frame by updating the data mask.

Warning: The algorithm seems to work, but isn't managing to update the DQ plane. I'm working on the issue. -MCW

Inputs and Outputs

The following options can be passed to rejectCosmicRays via the RecipeSystem, or overridden with a user configuration file, as marked:

Parameter	Type	Default	Override?		Description
			Recipe	User	
suffix	str	_cosmicRaysRejected	Y	Y	Suffix affixed to modified file.
subsampling	int	2	Y	Y	The number of pixels to subsample in each direction of each pixel.
sigma_lim	float	5.0	Y	Y	Sigma clipping value.
f_lim	float	6.0	Y	Y	Fine-structure clipping limit.
n_passes	int	2	Y	Y	Number of iterations to perform.

Algorithm

This primitive is a first-principles implementation of the LACosmic algorithm. NumPy array operations are used for maximum efficiency. For a full-discussion of the algorithm, see van Dokkum 2001, PASP **113**, 1420-1427 (ADS). In summary (where the symbol $*$ denotes convolution):

1. Create a Laplacian, $\mathcal{L}^+$, of the input data frame, I.
   - Subsample the image by a factor of `subsampling` and apply the Laplacian transformation;
   - Set any negative sub-pixels to have value 0 instead;
   - Re-sample the image back to its original resolution.
2. Generate the 'sigma map', S, of the frame.
   - Form a 'noise image' of the data, $N = g^{-1}\sqrt{g(M_5 * I) + \sigma_{rn}^2}$, where g and σ_{rn} are the data gain and read noise respectively. M_5 is a 5×5 median filter used to smooth the data.
   - The sigma map is then generated as $S = \mathcal{L}^+ / f_S N$, where f_S is the `subsampling` factor used.
   - Smooth the sigma map to help remove sampling flux, resulting in $S' = S - (S * M_5)$.
3. Generate the 'fine-structure image' of the frame, $\mathcal{F} = (M_3 * I) - ((M_3 * I) * M_7)$, where M_3 and M_7 are 3×3 and 7×7 median filters,
4. Mark pixels as cosmic rays (via the image quality plane) if:
   - $S' >$ `sigma_lim`, and
   - $\mathcal{L}^+ / \mathcal{F} >$ `f_lim`.
5. For the purposes of the algorithm (i.e. **not** in the data to be returned), replace cosmic rays pixels with the median value of surrounding pixels.
6. Iterate over steps 1-5 until `n_passes` is reached, or the number of cosmic rays detected falls to 0.

Issues and Limitations

Both strong spectral features and, in high-resolution mode, the ThXe calibration output may be incorrectly flagged as cosmic rays. Therefore, this primitive will trigger a call to **need to add more here once this actually works**

Note: The current preset values of `sigma_lim` (15.0, 5.0), `f_lim` (5.0) and `n_passes` (5) are a first-pass guess at what seems to flag roughly the correct number of cosmic rays. More complete testing, and the possible implementation of a table of preset values on a case-by-case basis, will need to wait until the findApertures routine (and related routines) are finalized and mixed in with the cosmic ray finding.

---

stackFrames

Purpose

This primitive is simply a wrapper around the standard `StackPrimitives.stackFrames` primitive which exposes extra parameters to the underlying `gemcombine.cl` IRAF script, and allows increasing the IRAF verbosity.

Inputs and Outputs

`hsigma`: float or None (default: None) - expose the hsigma parameter of the underlying IRAF gemcombine script, allowing it to be set within a recipe

`hthresh`: float or None (default: None) - expose the hthreshold parameter of the underlying IRAF gemcombine script, allowing it to be set within a recipe

`lsigma`: float or None (default: None) - expose the lsigma parameter of the underlying IRAF gemcombine script, allowing it to be set within a recipe

`lthresh`: float or None (default: None) - expose the lthreshold parameter of the underlying IRAF gemcombine script, allowing it to be set within a recipe

`pclip`: float or None (default: None) - expose the pclip parameter of the underlying IRAF gemcombine script, allowing it to be set within a recipe

`sigscale`: float or None (default: None) - expose the sigscale parameter of the underlying IRAF gemcombine script, allowing it to be set within a recipe

`verbose`: <any> or None (default: None) - set the level of iraf verbosity

Issues and Limitations

The default value for `reject_method` (a parameter exposed by the underlying `StackPrimitives.stackFrames` primitive) is 'avsigclip' (inherited from Gemini) which is probably not what we want in most cases. If necessary (or more convenient), we could override the default in this wrapper, or in the `parameters_GHOST.py` file.

standardizeHeaders

Purpose

This primitive implements part of the necessary interface required by the `StandardizePrimitives.prepare` "super-primitive" inherited from Gemini which, among other things, we use for its calls to `validateData` and `markAsPrepared`. Gemini defines both of those so we don't have to, but not `standardizeHeaders`; however, it does define `standardizeGeminiHeaders` which works fine for us so, ironically, we just make this primitive call that!

Inputs and Outputs

The usual stream of AstroData objects which, upon output, have had their headers standardized for trouble-free processing through the GHOST pipeline recipes.

Issues and Limitations

None.

standardizeStructure

Purpose

This primitive is responsible for massaging input data frames into a format compatible with the remainder of the downstream primitives.

When passed a series of slit viewer MEFs (such as results from the observation bundle splitter primitive, or from the simulator with the -split option sup- plied), it "promotes" all the slit viewer extensions therein to full AstroData objects with various manipulations so that they will pass various checks and otherwise work nicely with the rest of the recipe.

When passed other input (such as normal science detector frames) nothing is done except the addition of the obligatory timestamp keyword.

Inputs and Outputs

`standardizeStructure` takes no particular configuration inputs.

Algorithm

For slit viewer data, the `standardizeStructure` primitive performs these man- ipulations:

- It uses AstroData slicing (also known as "sub-data") to initially produce a separate AstroData object for each extension, then also makes a copy of the PHU for each resultant AstroData object (since sub-data references a single copy of the original PHU by default). In this manner, fully independent AstroData objects are created representing each extension.
- The above manipulation causes the resulting AstroData objects to all have the same ORIGNAME in their PHUs. This becomes an issue when we leverage the existing `StackPrimitives.stackFrames` primitive, as it must first (being IRAF-based) write its inputs to disk. As it uses ORIGNAME to do so, clashes would result, so we make ORIGNAME unique by simply appending an incrementing number.

- Finally, we also reset the EXTVER of all extensions in the AstroData objects resulting from the first manipulation back to 1 (since now there is only a single extension in each AstroData object).

Issues and Limitations

None.

switchTo

Purpose

This primitive makes the named stream the current stream, such that all subsequently invoked primitives will take as input the AstroData objects in that stream, as well as write their output AstroData objects to that stream.

Inputs and Outputs

The only parameter `switchTo` takes is `streamName`, the name of the stream to which to "shift" execution. Equivalently, the name of the stream which to make current.

Issues and Limitations

Be careful not to specify the parameter `streamName` as simply `stream` as this is a reserved parameter name with special meaning to the recipe framework and odd behaviour results when used.

tileAmplifiers

Purpose

This primitive tiles (or optionally mosaics) the SCI frames of the input images, along with the VAR and DQ frames if they exist.

Inputs and Outputs

`mosaic`: bool (default: False), tile the images by default; mosaic them (including transformations) if True

`dq_planes`: bool (default: False), transform the DQ image, bit plane by bit plane

Algorithm

This primitive is a copy of `GEMINIPrimitives.mosaicADdetectors` with a couple of additional hacks introduced just prior to calling the mosaicing routine, `gemini_mosaic_function`, and undone immediately after. These are necessary in order to trick the underlying mosaicing machinery into working for GHOST data, and are as follows:

- First, we add 'GSAOI' to the internal array of types for our input AstroData object (which contains the extensions we want to mosaic). This bypasses the GSAOI-/GMOS-only check built into `gemini_mosaic_function`. Afterwards we reset the type list back to normal by invoking `AstroData.refresh_types` on the resulting, mosaiced, AstroData object.
- Secondly, we set the `INSTRUME` PHU keyword to `../../../astrodata_GHOST/ADCONFIG_GHOST/lookups`. This tricks `set_geo_values` (called by `gemini_mosaic_function`) into referencing the `geometry_conf.py` file under our own `astrodata_GHOST/lookups` folder. As soon as the mosaicing routine returns we reset `INSTRUME` back to 'GHOST' (and in fact this must be done before the call to `refresh_types` mentioned above).

Until such time as `astrodata_GHOST` has been merged into `astrodata_Gemini` (like GMOS and GSAOI have been), or `astrodata_Gemini` is refactored, the above two hacks will be necessary in order for GHOST to make use of Gemini's mosaicing mechanism.

Issues and Limitations

Requires ugly hacks until integration with, or refactoring of, `astrodata_Gemini` is completed.

validateData

Purpose

This primitive is responsible for ensuring the data passed is GHOST data and has correctly formatted keywords. (We use the "prepare" superprimitive provided by Gemini which invokes validateData before standardizeHeaders and standardizeStructure, so validateData must not assert things set by either of those.)

Inputs and Outputs

`validateData` takes no particular configuration inputs.

Algorithm

None.

Issues and Limitations

At the moment, our version of this primitive does nothing other than write a timestamp into the headers. In future we may want to check that the right number of extensions is present (but only for normal science detector frames as slit viewer frames can have any number), and/or confirming that only certain binning shape is used. These are the sorts of assertions other instruments have used validateData for.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search