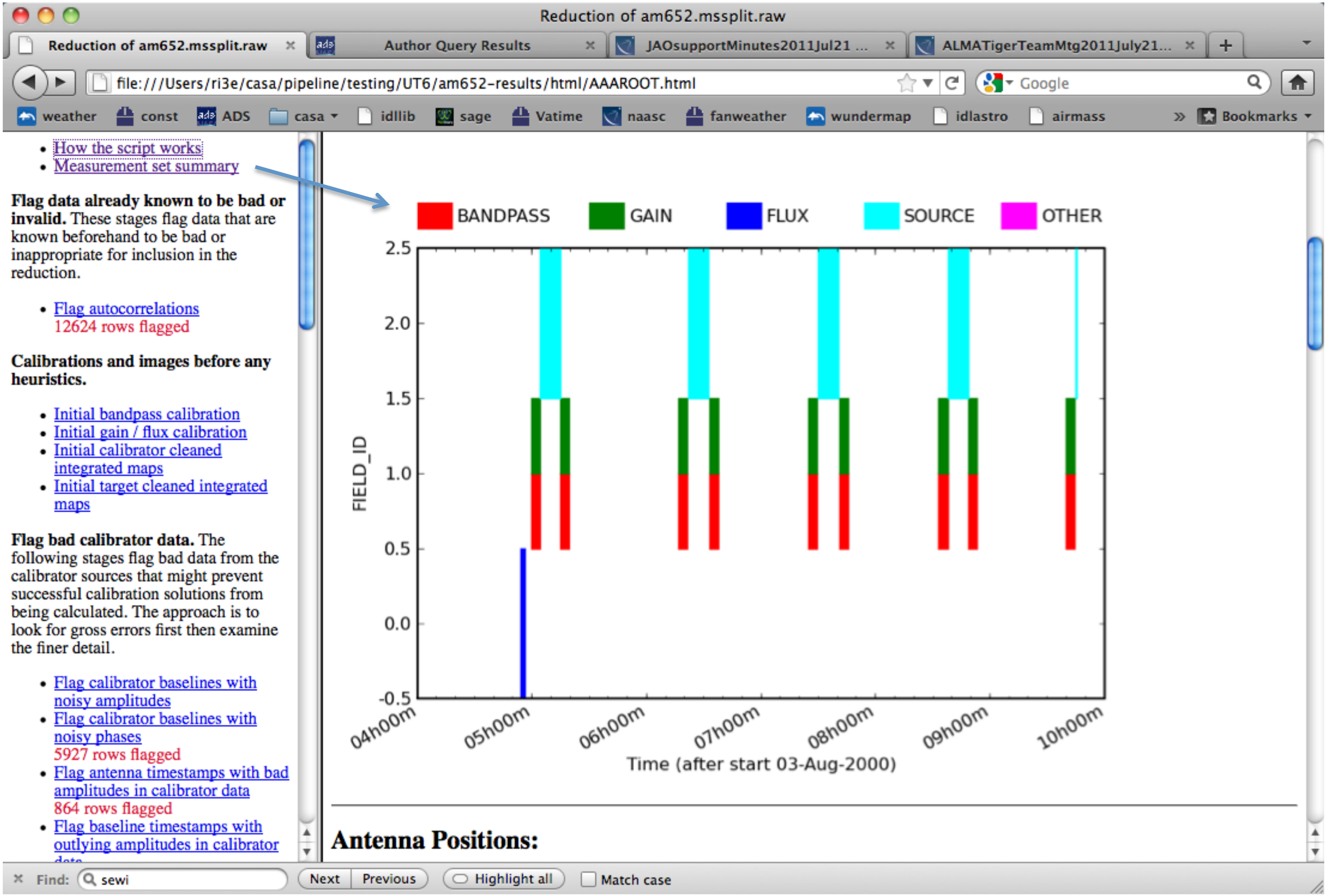
What the user sees:



- How the script works
- Measurement set summary

Flag data already known to be bad or invalid. These stages flag data that are known beforehand to be bad or inappropriate for inclusion in the reduction.

- Flag autocorrelations
  12624 rows flagged

Calibrations and images before any heuristics.

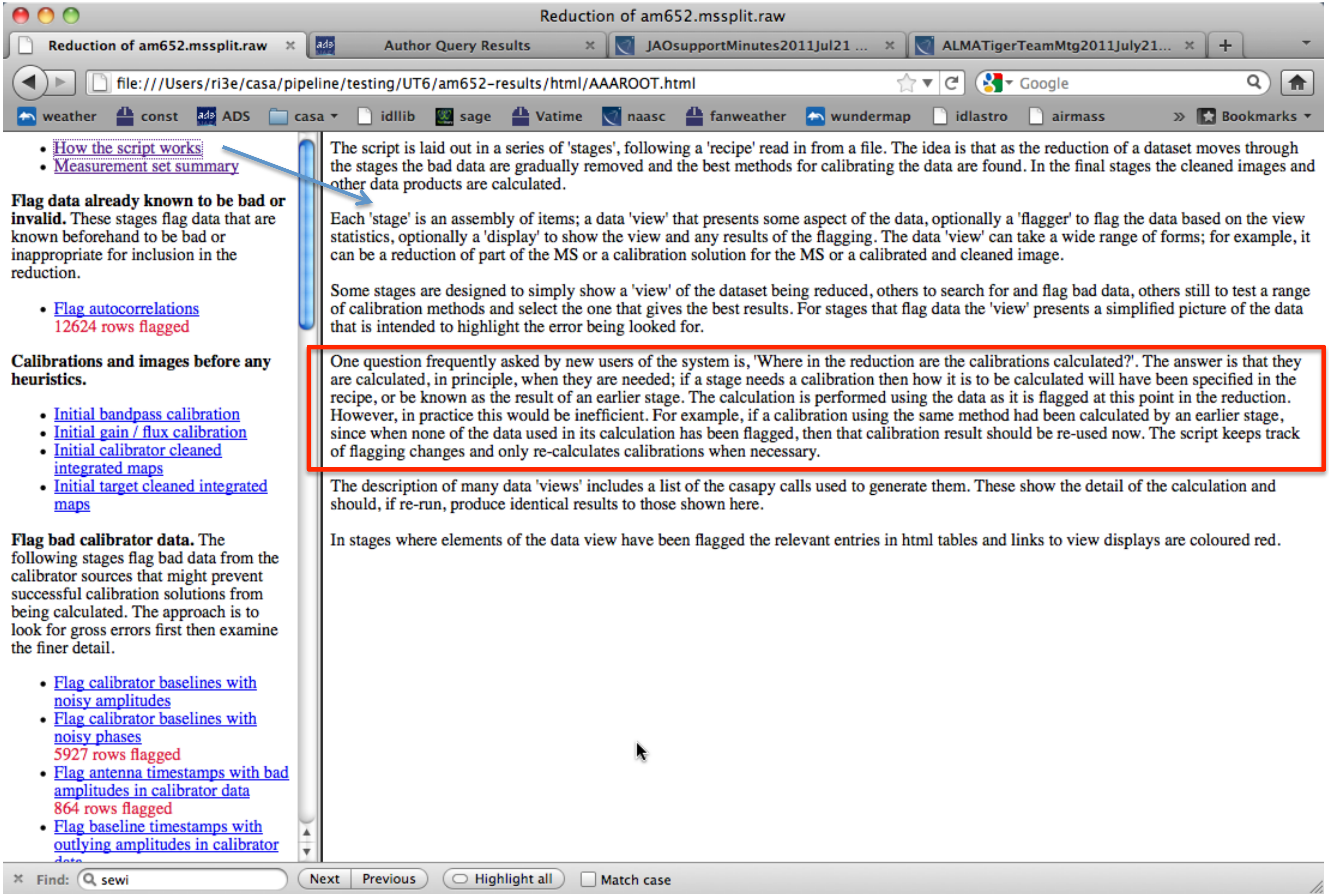
- Initial bandpass calibration
- Initial gain / flux calibration
- Initial calibrator cleaned integrated maps
- Initial target cleaned integrated maps

Flag bad calibrator data. The following stages flag bad data from the calibrator sources that might prevent successful calibration solutions from being calculated. The approach is to look for gross errors first then examine the finer detail.

- Flag calibrator baselines with noisy amplitudes
- Flag calibrator baselines with noisy phases
  5927 rows flagged
- Flag antenna timestamps with bad amplitudes in calibrator data
  864 rows flagged
- Flag baseline timestamps with outlying amplitudes in calibrator data

Antenna Positions:

What the user sees:


Reduction of am652.mssplit.raw
Reduction of am652.mssplit.raw
Author Query Results
JAOsupportMinutes2011Jul21 ...
ALMATigerTeamMtg2011July21...
file:///Users/ri3e/casa/pipeline/testing/UT6/am652-results/html/AAAROOT.html
Google
weather
const
ADS
casa
idllib
sage
Vatime
naasc
fanweather
wundermap
idlastro
airmass
Bookmarks
Find: sewi
Next
Previous
Highlight all
Match case

- How the script works
- Measurement set summary

Flag data already known to be bad or invalid. These stages flag data that are known beforehand to be bad or inappropriate for inclusion in the reduction.

- Flag autocorrelations
  12624 rows flagged

Calibrations and images before any heuristics.

- Initial bandpass calibration
- Initial gain / flux calibration
- Initial calibrator cleaned integrated maps
- Initial target cleaned integrated maps

Flag bad calibrator data. The following stages flag bad data from the calibrator sources that might prevent successful calibration solutions from being calculated. The approach is to look for gross errors first then examine the finer detail.

- Flag calibrator baselines with noisy amplitudes
- Flag calibrator baselines with noisy phases
  5927 rows flagged
- Flag antenna timestamps with bad amplitudes in calibrator data
  864 rows flagged
- Flag baseline timestamps with outlying amplitudes in calibrator data

The script is laid out in a series of 'stages', following a 'recipe' read in from a file. The idea is that as the reduction of a dataset moves through the stages the bad data are gradually removed and the best methods for calibrating the data are found. In the final stages the cleaned images and other data products are calculated.

Each 'stage' is an assembly of items; a data 'view' that presents some aspect of the data, optionally a 'flagger' to flag the data based on the view statistics, optionally a 'display' to show the view and any results of the flagging. The data 'view' can take a wide range of forms; for example, it can be a reduction of part of the MS or a calibration solution for the MS or a calibrated and cleaned image.

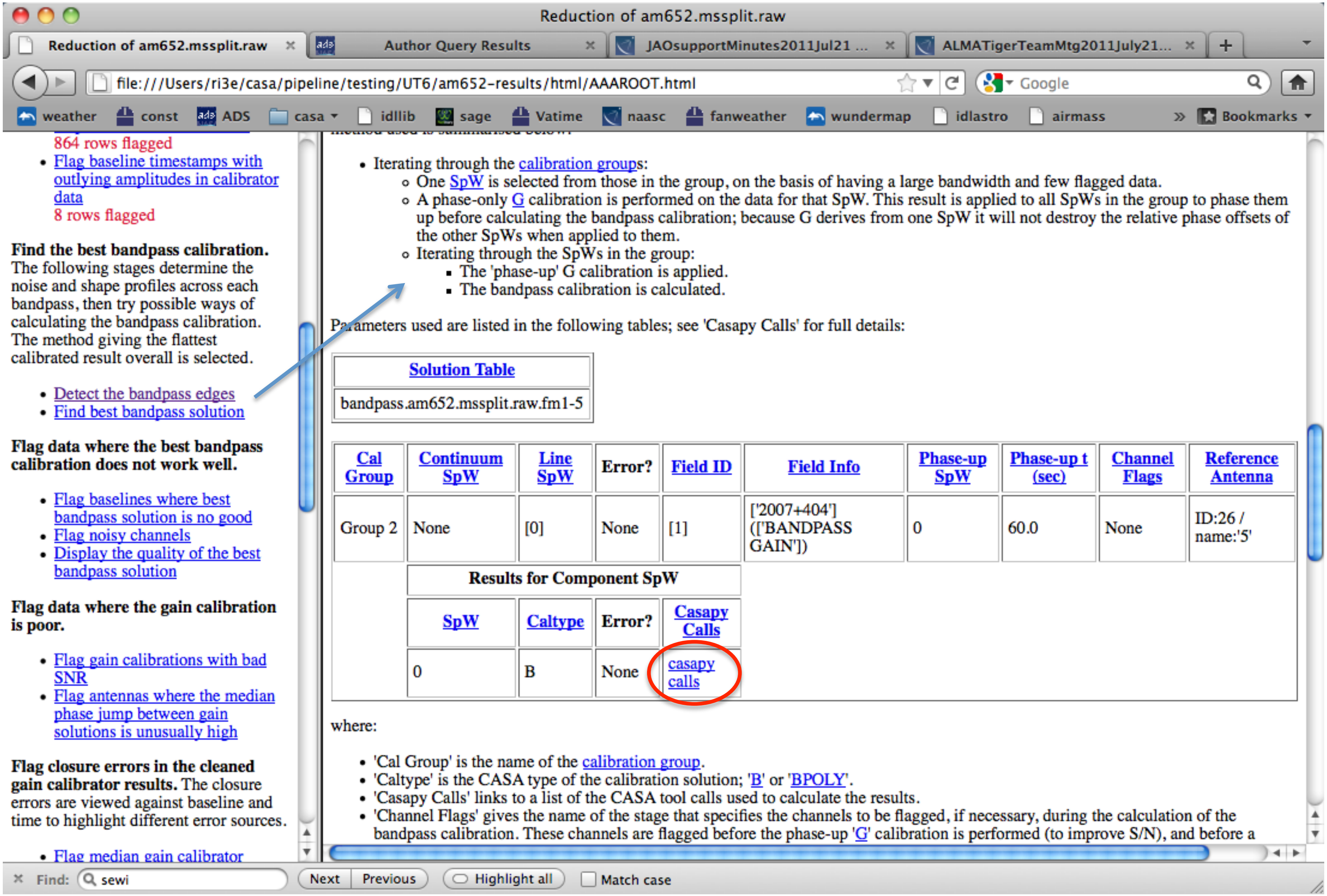
Some stages are designed to simply show a 'view' of the dataset being reduced, others to search for and flag bad data, others still to test a range of calibration methods and select the one that gives the best results. For stages that flag data the 'view' presents a simplified picture of the data that is intended to highlight the error being looked for.

One question frequently asked by new users of the system is, 'Where in the reduction are the calibrations calculated?'. The answer is that they are calculated, in principle, when they are needed; if a stage needs a calibration then how it is to be calculated will have been specified in the recipe, or be known as the result of an earlier stage. The calculation is performed using the data as it is flagged at this point in the reduction. However, in practice this would be inefficient. For example, if a calibration using the same method had been calculated by an earlier stage, since when none of the data used in its calculation has been flagged, then that calibration result should be re-used now. The script keeps track of flagging changes and only re-calculates calibrations when necessary.

The description of many data 'views' includes a list of the casapy calls used to generate them. These show the detail of the calculation and should, if re-run, produce identical results to those shown here.

In stages where elements of the data view have been flagged the relevant entries in html tables and links to view displays are coloured red.

If you want to "run" the pipeline


Reduction of am652.mssplit.raw
Reduction of am652.mssplit.raw
Author Query Results
JAOsupportMinutes2011Jul21 ...
ALMATigerTeamMtg2011July21...
file:///Users/ri3e/casa/pipeline/testing/UT6/am652-results/html/AAAROOT.html
Google
weather
const
ADS
casa
idllib
sage
Vatime
naasc
fanweather
wundermap
idlastro
airmass
Bookmarks
Find: sewi
Next
Previous
Highlight all
Match case

864 rows flagged

- Flag baseline timestamps with outlying amplitudes in calibrator data
  8 rows flagged

Find the best bandpass calibration. The following stages determine the noise and shape profiles across each bandpass, then try possible ways of calculating the bandpass calibration. The method giving the flattest calibrated result overall is selected.

- Detect the bandpass edges
- Find best bandpass solution

Flag data where the best bandpass calibration does not work well.

- Flag baselines where best bandpass solution is no good
- Flag noisy channels
- Display the quality of the best bandpass solution

Flag data where the gain calibration is poor.

- Flag gain calibrations with bad SNR
- Flag antennas where the median phase jump between gain solutions is unusually high

Flag closure errors in the cleaned gain calibrator results. The closure errors are viewed against baseline and time to highlight different error sources.

- Flag median gain calibrator

- Iterating through the calibration groups:
  - One SpW is selected from those in the group, on the basis of having a large bandwidth and few flagged data.
  - A phase-only G calibration is performed on the data for that SpW. This result is applied to all SpWs in the group to phase them up before calculating the bandpass calibration; because G derives from one SpW it will not destroy the relative phase offsets of the other SpWs when applied to them.
  - Iterating through the SpWs in the group:
    - The 'phase-up' G calibration is applied.
    - The bandpass calibration is calculated.

Parameters used are listed in the following tables; see 'Casapy Calls' for full details:

Solution Table
bandpass.am652.mssplit.raw.fm1-5

Cal Group	Continuum SpW	Line SpW	Error?	Field ID	Field Info	Phase-up SpW	Phase-up t (sec)	Channel Flags	Reference Antenna
Group 2	None	[0]	None	[1]	['2007+404'] (['BANDPASS GAIN'])	0	60.0	None	ID:26 / name:'5'

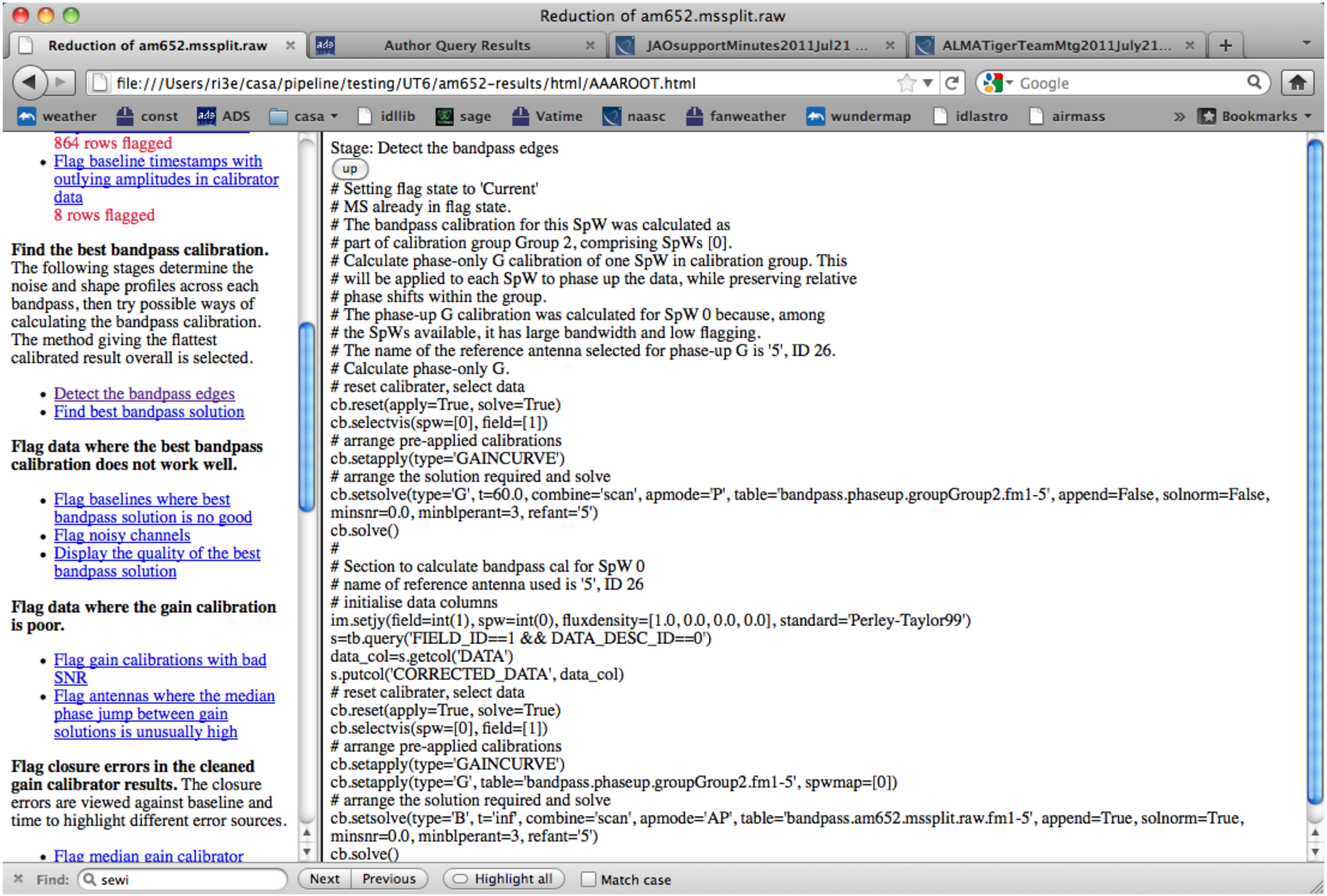
Results for Component SpW

SpW	Caltype	Error?	Casapy Calls
0	B	None	casapy calls

where:

- 'Cal Group' is the name of the calibration group.
- 'Caltype' is the CASA type of the calibration solution; 'B' or 'BPOLY'.
- 'Casapy Calls' links to a list of the CASA tool calls used to calculate the results.
- 'Channel Flags' gives the name of the stage that specifies the channels to be flagged, if necessary, during the calculation of the bandpass calibration. These channels are flagged before the phase-up 'G' calibration is performed (to improve S/N), and before a

If you want to "run" the pipeline


Reduction of am652.mssplit.raw
Reduction of am652.mssplit.raw
Author Query Results
JAOsupportMinutes2011Jul21 ...
ALMATigerTeamMtg2011July21...
file:///Users/ri3e/casa/pipeline/testing/UT6/am652-results/html/AAAROOT.html
Google
weather
const
ADS
casa
idllib
sage
Vatime
naasc
fanweather
wundermap
idlastro
airmass
Bookmarks
Find:
sewi
Next
Previous
Highlight all
Match case

864 rows flagged

- Flag baseline timestamps with outlying amplitudes in calibrator data
  8 rows flagged

Find the best bandpass calibration. The following stages determine the noise and shape profiles across each bandpass, then try possible ways of calculating the bandpass calibration. The method giving the flattest calibrated result overall is selected.

- Detect the bandpass edges
- Find best bandpass solution

Flag data where the best bandpass calibration does not work well.

- Flag baselines where best bandpass solution is no good
- Flag noisy channels
- Display the quality of the best bandpass solution

Flag data where the gain calibration is poor.

- Flag gain calibrations with bad SNR
- Flag antennas where the median phase jump between gain solutions is unusually high

Flag closure errors in the cleaned gain calibrator results. The closure errors are viewed against baseline and time to highlight different error sources.

- Flag median gain calibrator

Stage: Detect the bandpass edges

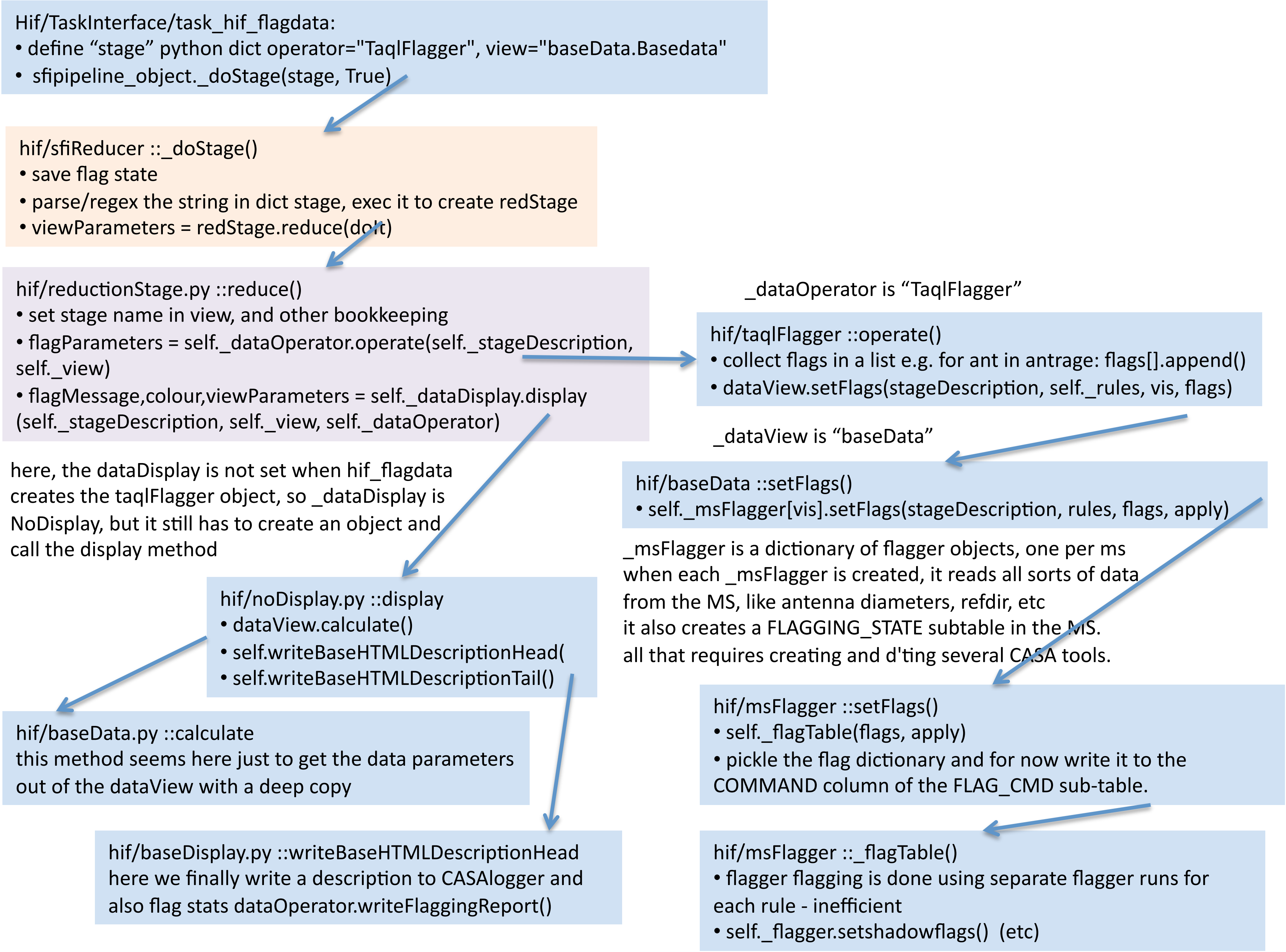
up

```
# Setting flag state to 'Current'
# MS already in flag state.
# The bandpass calibration for this SpW was calculated as
# part of calibration group Group 2, comprising SpWs [0].
# Calculate phase-only G calibration of one SpW in calibration group. This
# will be applied to each SpW to phase up the data, while preserving relative
# phase shifts within the group.
# The phase-up G calibration was calculated for SpW 0 because, among
# the SpWs available, it has large bandwidth and low flagging.
# The name of the reference antenna selected for phase-up G is '5', ID 26.
# Calculate phase-only G.
# reset calibrater, select data
cb.reset(apply=True, solve=True)
cb.selectvis(spw=[0], field=[1])
# arrange pre-applied calibrations
cb.setapply(type='GAINCURVE')
# arrange the solution required and solve
cb.setsolve(type='G', t=60.0, combine='scan', apmode='P', table='bandpass.phaseup.groupGroup2.fm1-5', append=False, solnorm=False,
minsnr=0.0, minblperant=3, refant='5')
cb.solve()
#
# Section to calculate bandpass cal for SpW 0
# name of reference antenna used is '5', ID 26
# initialise data columns
im.setjy(field=int(1), spw=int(0), fluxdensity=[1.0, 0.0, 0.0, 0.0], standard='Perley-Taylor99')
s=tb.query('FIELD_ID==1 && DATA_DESC_ID==0')
data_col=s.getcol('DATA')
s.putcol('CORRECTED_DATA', data_col)
# reset calibrater, select data
cb.reset(apply=True, solve=True)
cb.selectvis(spw=[0], field=[1])
# arrange pre-applied calibrations
cb.setapply(type='GAINCURVE')
cb.setapply(type='G', table='bandpass.phaseup.groupGroup2.fm1-5', spwmap=[0])
# arrange the solution required and solve
cb.setsolve(type='B', t='inf', combine='scan', apmode='AP', table='bandpass.am652.mssplit.raw.fm1-5', append=True, solnorm=True,
minsnr=0.0, minblperant=3, refant='5')
cb.solve()
```

Pipeline Implementation, Remy's understanding (or lack thereof)
July 2011

	Example	What it is	What it does
recipe	alma_ld_recipe.py	function hreduce	calls a series of tasks
task	hif_bandpass.py	function	calls the reducer to do execute a stage, controlled by a long string including reduction stage and display
pipeline object	sfiReducer.py	class	parses the string, and calls operator.operate() and display.display()
operator	bandpassEdgeFlagger.py	class	e.g. _flagData (which doesn't flag ☺)
display	sliceDisplay.py	class	view::getdata(), create html and matplots
view	bandpassCalibration.py	class w/ memory of ms state	bookkeeping, call e.g. calibrator tool, write out CASA commands

operator is null (for everything but flagging), but
view::getdata() can contain a lot of actual operations, e.g. for bandpass
that's where the bandpass actually gets calculated



Hif/TaskInterface/task_hif_flagdata:

- define “stage” python dict operator="TaqlFlagger", view="baseData.Basedata"
- sfipipeline_object._doStage(stage, True)

hif/sfiReducer ::_doStage()

- save flag state
- parse/regex the string in dict stage, exec it to create redStage
- viewParameters = redStage.reduce(doIt)

hif/reductionStage.py ::reduce()

- set stage name in view, and other bookkeeping
- flagParameters = self._dataOperator.operate(self._stageDescription, self._view)
- flagMessage,colour,viewParameters = self._dataDisplay.display (self._stageDescription, self._view, self._dataOperator)

here, the dataDisplay is not set when hif_flagdata creates the taqlFlagger object, so _dataDisplay is NoDisplay, but it still has to create an object and call the display method

hif/noDisplay.py ::display

- dataView.calculate()
- self.writeBaseHTMLDescriptionHead(
- self.writeBaseHTMLDescriptionTail()

hif/baseData.py ::calculate
this method seems here just to get the data parameters out of the dataView with a deep copy

hif/baseDisplay.py ::writeBaseHTMLDescriptionHead
here we finally write a description to CASAlogger and also flag stats dataOperator.writeFlaggingReport()

_dataOperator is “TaqlFlagger”

hif/taqlFlagger ::operate()

- collect flags in a list e.g. for ant in antrage: flags[].append()
- dataView.setFlags(stageDescription, self._rules, vis, flags)

_dataView is “baseData”

hif/baseData ::setFlags()

- self._msFlagger[vis].setFlags(stageDescription, rules, flags, apply)

_msFlagger is a dictionary of flagger objects, one per ms
when each _msFlagger is created, it reads all sorts of data from the MS, like antenna diameters, refdir, etc
it also creates a FLAGGING_STATE subtable in the MS.
all that requires creating and d'ting several CASA tools.

hif/msFlagger ::setFlags()

- self._flagTable(flags, apply)
- pickle the flag dictionary and for now write it to the COMMAND column of the FLAG_CMD sub-table.

hif/msFlagger ::_flagTable()

- flagger flagging is done using separate flagger runs for each rule - inefficient
- self._flagger.setshadowflags() (etc)

(the start of) their ALMA recipe

- task_hif_flagdata:
    - operator taqlFlagger::operate()
        - iterate flag commands and collect lists of flags to apply, call dataView::setFlags
    - view baseData::setFlags()
        - self._msFlagger[vis].setFlags()
            - self._flagger.setshadowflags()   etc   [_flagger is the CASA tool]
    - display noDisplay::display(): dataView.calculate()
    - baseData::calculate(): deepcopy view parameters to display class

- hif_delaycal:
    - operator = null, so nothing happens until _display::display()
    - bandpassCal = “bandpassCalibration, NoDisplay”
    - view delayCalibration
    - display sliceDisplay
        - delayCalibration::getData
            - bandpassCalibration::calculate() – if there’s one avail in bookKeeper then just return it, else actually calculate, bpoly for each spw, bchan for spw grp
            - read BP gain, calculate median phase jump per channel

- hif_findedgechan
    - operator = BandpassEdgeFlagger
        - ::operate(): dataView.getData()
            - medianAndMad modifies the BandpassCalibration view, so
            - bandpassCalibration::getData()
                - calculate THREE bps flag states “current” “original” “stageEntry”
                    - init scratchcols with setJy
                    - apply vla gain and delay
                    - calc phase-only G
                    - calc bpolys for each spw in group
                    - calc bchan for group
                - read cal tables directly and try to figure out SPW maps
                - query caltables with TaQl query
                - calculate amp, phase of gains in python,
                - average poln’s in python if requested
        - collect flagged chans from each method and store in the dataView for later (doesn’t actually ever flag anything)
    - view = MedianAndMAD(view=BandpassCalibration)
    - display =sliceDisplay.SliceX()
        - Display::display does another dataView::getData() – does that mean it calculates all the stats etc _again_?

- hif_sfcalclean
    - operator=null
    - display=SkyDisplay
    - view=CleanImageV3 (bgCal=none, gainCal=none)
        - getData()
            - self.calculate()
                - force calculation of a new BP and Gain (I think… at last a rewrapping of them – unclear whether it uses existing one from a different stage)
            - commands += BaseImage._fillData() for each image
            - start up an ia and imager and regionmanager tool if not already started
            - apply delay, bp, gain
            - make taql to combine images
            - selectvis
            - set imaging weights
            - get cell and imsize from advise()
            - makeimage(psf), fitpsf,
            - “pilot” mfs
            - iterated masking with Amy’s heuristics (I think)
    - skydisplay does the multiple panels

Their Full recipe for ALMA

- flagdata: autocorr, shadow
- delaycal (implicit BP with phaseup)
- findedgechan (implicit BP): “mark” 5% edges, but don’t flag, just exclude from plots later
- sfcalclean: (implicit BP with phaseup, gaincal) mfs image calibrators
- sfcalclean: (apply BP, gaincal) mfs image target
- flag cals raw 7σ amplitudes (ampnoise_bim: baseline-based, chan and time medianed)
- flag cals raw 7σ phases (phasenoise_bim: baseline-based, chan and time medianed)
- flag gain soln 7σ amplitudes (ampgain_atim: ant-based, chan medianed, fn of time)
- flag cals raw 7σ amplitudes (ampnoise_btim: baseline-based, chan medianed, fn of time)

- sfcalclean: (implicit new BP w/phaseup and gaincal) mfs image phasecals for each MS
- sfcalclean: (implicit new BP w/phaseup and gaincal) mfs image target for each MS
- sfcalclean: mfs image phasecal for combined MSs (not sure if BP,gain are combined
- sfcaldirty: mfs dirty image of target
- findcubelines (presumably, there’s an implicit dirty cube inversion here)
- sfcalclean: clean target cubes over chan ranges where there are lines
- sfcalclean: mfs target image of continuum chans
- contsub: clean target cubes with continuum subtracted
- findlines: extract spectra from target images

Our Recipe for ALMA: TBD

- flagdata: autocorr, shadow
- ~~delaycal (implicit BP with phaseup)~~
- findedgechan (implicit BP): flag 5% edges
- ~~sfcalclean: (implicit BP with phaseup, gaincal) mfs image calibrators~~
- ~~sfcalclean: (apply BP, gaincal) mfs image target~~
- flag cals raw 7σ amplitudes (ampnoise_bim: baseline-based, chan and time medianed)
- ~~flag cals raw 7σ phases (phasenoise_bim: baseline-based, chan and time medianed)~~
- flag gain soln 7σ amplitudes (ampgain_atim: ant-based, chan medianed, fn of time)
- flag cals raw 7σ amplitudes (ampnoise_btim: baseline-based, chan medianed, fn of time)

Explicit BP, show phaseup of BP and flag that gain soln

- sfcalclean: (implicit new BP w/phaseup and gaincal) mfs image phasecals for each MS
- sfcalclean: (implicit new BP w/phaseup and gaincal) mfs image target for each MS
- sfcalclean: mfs image phasecal for combined MSs (not sure if BP,gain are combined
- sfcaldirty: mfs dirty image of target
- findcubelines (presumably, there's an implicit dirty cube inversion here)
- sfcalclean: clean target cubes over chan ranges where there are lines
- sfcalclean: mfs target image of continuum chans
- contsub: clean target cubes with continuum subtracted
- findlines: extract spectra from target images

Issues to consider

- ❑ Processing Design
  - ❑ Can the concept of “regenerate calibrations and images on the fly whenever flagging or other changes require them” result in a clear reduction path?
  - ❑ How can we make the recipe/stages more clearly show the reduction path to the user?
  - ❑ Are the generated python scripts sufficient to give the user? (i.e. they’ll NEVER “run the pipeline”)
  - ❑ Is the design viable, especially can it be made efficient?
    - ❑ inefficient steps include the creation/destruction of tools, generation of python exec strings

- ❑ Tree-structure html output design
  - ❑ How can it be made more readable? The actual steps clearer? (⬆)
  - ❑ How can the user recreate some of the plots and diagnostic output? (right now they are given the CASA tool flag/cal/etc commands, but not any plotms or plotcal commands)
  - ❑ Can a concise summary be generated simultaneously with the html.tgz?