

---

# Pipeline Tasks Reference Manual

*Release 2025.1.0*

**pipeline team**

Dec 10, 2025

## Pipeline Tasks

<b>1 pipeline.h.cli</b>	<b>2</b>
1.1 pipeline.h.cli.h_init . . . . .	2
1.2 pipeline.h.cli.h_resume . . . . .	3
1.3 pipeline.h.cli.h_save . . . . .	3
1.4 pipeline.h.cli.h_tsyscal . . . . .	3
1.5 pipeline.h.cli.h_weblog . . . . .	4
<b>2 pipeline.hif.cli</b>	<b>5</b>
2.1 pipeline.hif.cli.hif_analyzealpha . . . . .	5
2.2 pipeline.hif.cli.hif_applycal . . . . .	6
2.3 pipeline.hif.cli.hif_checkproductsizes . . . . .	7
2.4 pipeline.hif.cli.hif_correctedampflag . . . . .	8
2.5 pipeline.hif.cli.hif_editimlist . . . . .	9
2.6 pipeline.hif.cli.hif_findcont . . . . .	11
2.7 pipeline.hif.cli.hif_lowgainflag . . . . .	12
2.8 pipeline.hif.cli.hif_makecutoutimages . . . . .	13
2.9 pipeline.hif.cli.hif_makeimages . . . . .	13
2.10 pipeline.hif.cli.hif_makeimlist . . . . .	15
2.11 pipeline.hif.cli.hif_makermssimages . . . . .	18
2.12 pipeline.hif.cli.hif_mstransform . . . . .	18
2.13 pipeline.hif.cli.hif_rawflagchans . . . . .	19
2.14 pipeline.hif.cli.hif_refant . . . . .	21
2.15 pipeline.hif.cli.hif_selfcal . . . . .	22
2.16 pipeline.hif.cli.hif_setjy . . . . .	24
2.17 pipeline.hif.cli.hif_setmodels . . . . .	25
2.18 pipeline.hif.cli.hif_transformimagedata . . . . .	26
2.19 pipeline.hif.cli.hif_uvcontsub . . . . .	27
<b>3 pipeline.hifa.cli</b>	<b>29</b>
3.1 pipeline.hifa.cli.hifa_antpos . . . . .	30
3.2 pipeline.hifa.cli.hifa_bandpass . . . . .	31
3.3 pipeline.hifa.cli.hifa_bandpassflag . . . . .	34
3.4 pipeline.hifa.cli.hifa_bpsolint . . . . .	37
3.5 pipeline.hifa.cli.hifa_diffgaincal . . . . .	39
3.6 pipeline.hifa.cli.hifa_exportdata . . . . .	41
3.7 pipeline.hifa.cli.hifa_flagdata . . . . .	42
3.8 pipeline.hifa.cli.hifa_flagtargets . . . . .	44
3.9 pipeline.hifa.cli.hifa_fluxcalflag . . . . .	45
3.10 pipeline.hifa.cli.hifa_gaincalsnr . . . . .	45
3.11 pipeline.hifa.cli.hifa_gfluxscale . . . . .	47

3.12	<code>pipeline.hifa.cli.hifa_gfluxscaleflag</code>	49
3.13	<code>pipeline.hifa.cli.hifa_imageprecheck</code>	50
3.14	<code>pipeline.hifa.cli.hifa_importdata</code>	51
3.15	<code>pipeline.hifa.cli.hifa_lock_refant</code>	53
3.16	<code>pipeline.hifa.cli.hifa_polcal</code>	53
3.17	<code>pipeline.hifa.cli.hifa_polcalflag</code>	54
3.18	<code>pipeline.hifa.cli.hifa_renorm</code>	54
3.19	<code>pipeline.hifa.cli.hifa_restoredata</code>	56
3.20	<code>pipeline.hifa.cli.hifa_session_refant</code>	57
3.21	<code>pipeline.hifa.cli.hifa_spwphaseup</code>	58
3.22	<code>pipeline.hifa.cli.hifa_targetflag</code>	61
3.23	<code>pipeline.hifa.cli.hifa_timegaincal</code>	62
3.24	<code>pipeline.hifa.cli.hifa_tsysflag</code>	64
3.25	<code>pipeline.hifa.cli.hifa_tsysflagcontamination</code>	66
3.26	<code>pipeline.hifa.cli.hifa_unlock_refant</code>	67
3.27	<code>pipeline.hifa.cli.hifa_wvrgcal</code>	67
3.28	<code>pipeline.hifa.cli.hifa_wvrgcalflag</code>	69
<b>4</b>	<b><code>pipeline.hifv.cli</code></b>	<b>72</b>
4.1	<code>pipeline.hifv.cli.hifv_analyzestokescubes</code>	73
4.2	<code>pipeline.hifv.cli.hifv_applycals</code>	73
4.3	<code>pipeline.hifv.cli.hifv_checkflag</code>	74
4.4	<code>pipeline.hifv.cli.hifv_circfeedpolcal</code>	75
4.5	<code>pipeline.hifv.cli.hifv_exportdata</code>	76
4.6	<code>pipeline.hifv.cli.hifv_exportvlassdata</code>	77
4.7	<code>pipeline.hifv.cli.hifv_finalcals</code>	77
4.8	<code>pipeline.hifv.cli.hifv_fixpointing</code>	78
4.9	<code>pipeline.hifv.cli.hifv_flagcal</code>	78
4.10	<code>pipeline.hifv.cli.hifv_flagdata</code>	79
4.11	<code>pipeline.hifv.cli.hifv_flagtargetsdata</code>	80
4.12	<code>pipeline.hifv.cli.hifv_fluxboot</code>	80
4.13	<code>pipeline.hifv.cli.hifv_hanning</code>	81
4.14	<code>pipeline.hifv.cli.hifv_importdata</code>	82
4.15	<code>pipeline.hifv.cli.hifv_mstransform</code>	83
4.16	<code>pipeline.hifv.cli.hifv_pbcor</code>	85
4.17	<code>pipeline.hifv.cli.hifv_plotsummary</code>	85
4.18	<code>pipeline.hifv.cli.hifv_priorcals</code>	85
4.19	<code>pipeline.hifv.cli.hifv_restoredata</code>	86
4.20	<code>pipeline.hifv.cli.hifv_restorepims</code>	88
4.21	<code>pipeline.hifv.cli.hifv_selfcal</code>	88
4.22	<code>pipeline.hifv.cli.hifv_semiFinalBPdcals</code>	89
4.23	<code>pipeline.hifv.cli.hifv_solist</code>	89
4.24	<code>pipeline.hifv.cli.hifv_statwt</code>	90
4.25	<code>pipeline.hifv.cli.hifv_syspower</code>	90
4.26	<code>pipeline.hifv.cli.hifv_testBPdcals</code>	91
4.27	<code>pipeline.hifv.cli.hifv_vlasetjy</code>	92
4.28	<code>pipeline.hifv.cli.hifv_vlassmasking</code>	92
<b>5</b>	<b><code>pipeline.hsd.cli</code></b>	<b>94</b>
5.1	<code>pipeline.hsd.cli.hsd_applycal</code>	94
5.2	<code>pipeline.hsd.cli.hsd_atmcor</code>	95
5.3	<code>pipeline.hsd.cli.hsd_baseline</code>	97
5.4	<code>pipeline.hsd.cli.hsd_bfflag</code>	101
5.5	<code>pipeline.hsd.cli.hsd_exportdata</code>	103

5.6	pipeline.hsd.cli.hsd_flagdata . . . . .	103
5.7	pipeline.hsd.cli.hsd_imaging . . . . .	105
5.8	pipeline.hsd.cli.hsd_importdata . . . . .	106
5.9	pipeline.hsd.cli.hsd_k2jycal . . . . .	108
5.10	pipeline.hsd.cli.hsd_restoredata . . . . .	109
5.11	pipeline.hsd.cli.hsd_skycal . . . . .	111
5.12	pipeline.hsd.cli.hsd_tsysflag . . . . .	113
<b>6</b>	<b>pipeline.hsdn.cli</b> . . . . .	<b>116</b>
6.1	pipeline.hsdn.cli.hsdn_exportdata . . . . .	116
6.2	pipeline.hsdn.cli.hsdn_importdata . . . . .	117
6.3	pipeline.hsdn.cli.hsdn_restoredata . . . . .	118
<b>Python Module Index</b>		<b>121</b>
<b>Index</b>		<b>122</b>

<i>h.cli</i>	Generic Tasks
<i>hif.cli</i>	Interferometry Generic Tasks
<i>hifa.cli</i>	Interferometry ALMA Tasks
<i>hifv.cli</i>	Interferometry VLA Tasks
<i>hsd.cli</i>	Single Dish ALMA Tasks
<i>hsdn.cli</i>	Single Dish Nobeyama Tasks

## PIPELINE.H.CLI

Generic Tasks

### Functions

<code>h_init</code>	Initialize the pipeline state and context.
<code>h_resume</code>	Restore a saved pipeline state.
<code>h_save</code>	Save the current pipeline state to disk.
<code>h_tsyscal</code>	Derive Tsys calibration tables for a list of ALMA MeasurementSets.
<code>h_weblog</code>	Open the pipeline weblog in a browser tab or window.

## 1.1 pipeline.h.cli.h\_init

`h_init(loglevel='info', plotlevel='default', weblog=True, processing_intents=None)`

Initialize the pipeline state and context.

`h_init` must be called before any other pipeline task. The pipeline can be initialized in one of two ways: by creating a new pipeline state (`h_init`) or be loading a saved pipeline state (`h_resume`).

`h_init` creates an empty pipeline context but does not load visibility data into the context. Any of the pipeline `h*_importdata` tasks can be used to load data.

### Parameters

- **loglevel** -- Log level for pipeline messages. Log messages below this threshold will not be displayed.
- **plotlevel** -- Toggle generation of detail plots in the web log. A level of 'all' generates all plots; 'summary' omits detail plots; 'default' generates all plots apart from for the hif\_applycal task.
- **weblog** -- Generate the web log
- **processing\_intents** -- Dictionary of processing intents for the current pipeline run.

### Returns

The results object for the pipeline task is returned.

### Examples

1. Create the pipeline context

```
>>> h_init()
```

## 1.2 pipeline.h.cli.h\_resume

**h\_resume**(*filename*: str | None = None)

Restore a saved pipeline state.

Restores a named pipeline state from disk, allowing a suspended pipeline reduction session to be resumed.

### Parameters

**filename** -- Saved pipeline state name. If set to 'last' or left as **None**, the most recently saved state ending with '.context' will be restored.

### Returns

The pipeline *context* object.

### Examples

Resume the last saved session:

```
>>> h_resume()
```

Resume from a saved session using the *context* snapshot after the processing stage-35:

```
>>> h_resume(filename='pipeline-20230227T202157/saved_state/context-stage35.pickle')
```

## 1.3 pipeline.h.cli.h\_save

**h\_save**(*filename*: str | None = None) → None

Save the current pipeline state to disk.

If no filename is given, the name of the pipeline *context* object will be used. This name typically consists of the pipeline procedure name (or 'pipeline-' plus a timestamp) with the suffix '.context'.

### Parameters

**filename** -- Optional target filename for saving the pipeline state.

### Returns

None

### Examples

Save the current state to a default file:

```
>>> h_save()
```

Save the current state to a file named 'savestate\_1':

```
>>> h_save(filename='savestate_1')
```

## 1.4 pipeline.h.cli.h\_tsyscal

**h\_tsyscal**(*vis=None*, *caltable=None*, *chantol=None*, *parallel=None*) → ResultsList[TsyscalResults]

Derive Tsys calibration tables for a list of ALMA MeasurementSets.

### Parameters

- **vis** -- List of input visibility files.

Example: `vis=['ngc5921.ms']`

- **caltable** -- Name of output gain calibration tables.

Example: `caltable='ngc5921.gcal'`

- **chantol** -- The tolerance in channels for mapping atmospheric calibration windows (TDM) to science windows (FDM or TDM).

Example: `chantol=5`

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: `'automatic', 'true', 'false', True, False`

Default: `None` (equivalent to `False`)

#### Returns

The results object for the pipeline task is returned.

### Examples

1. Standard call

```
>>> h_tsyscal()
```

## 1.5 pipeline.h.cli.h\_weblog

### `h_weblog(relpath=None)`

Open the pipeline weblog in a browser tab or window.

#### Parameters

**relpath** -- Relative path to the weblog index file. This file must be located in a child directory of the CASA working directory. If relpath is left unspecified, the most recent weblog will be located and displayed.

#### Returns

None

### Examples

1. Open pipeline weblog in a browser:

```
>>> h_weblog()
```

## PIPELINE.HIF.CLI

Interferometry Generic Tasks

### Functions

<code>hif_analyzealpha</code>	Extract spectral index from intensity peak in VLASS images.
<code>hif_applycal</code>	Apply precomputed calibrations to the data.
<code>hif_checkproductsize</code>	Check imaging product size.
<code>hif_correctedampflag</code>	Flag corrected - model amplitudes based on calibrators.
<code>hif_editimlist</code>	Add to a list of images to be produced with <code>hif_makeimages</code> .
<code>hif_findcont</code>	Find continuum frequency ranges for a list of specified targets.
<code>hif_lowgainflag</code>	Flag antennas with low or high gain.
<code>hif_makecutoutimages</code>	Cutout central 1 sq.
<code>hif_makeimages</code>	Compute clean results from a list of specified targets.
<code>hif_makeimlist</code>	Compute list of clean images to be produced.
<code>hif_makermssimages</code>	Create RMS images for VLASS data.
<code>hif_mstransform</code>	Create new MeasurementSets for science target imaging.
<code>hif_rawflagchans</code>	Flag deviant baseline/channels in raw data.
<code>hif_refant</code>	Select the best reference antennas.
<code>hif_selfcal</code>	Determine and apply self-calibration with the science target data.
<code>hif_setjy</code>	Fill the model column with calibrated visibilities.
<code>hif_setmodels</code>	Set calibrator source models.
<code>hif_transformimagedata</code>	Extract fields for the desired VLASS image to a new MS and reset weights if desired.
<code>hif_uvcontsub</code>	Fit and subtract continuum from the data.

### 2.1 pipeline.hif.cli.hif\_analyzealpha

`hif_analyzealpha(image=None, alphafile=None, alphaerrorfile=None)` → Results

Extract spectral index from intensity peak in VLASS images.

#### Parameters

- `image` -- Restored subimage
- `alphafile` -- Input spectral index map
- `alphaerrorfile` -- Input spectral index error map

#### Returns

The results object for the pipeline task is returned.

## Examples

1. Basic analyzealpha task

```
>>> hif_analyzealpha()
```

## 2.2 pipeline.hif.cli.hif\_applycal

**hif\_applycal**(vis=None, field=None, intent=None, spw=None, antenna=None, parang=None, applymode=None, calwt=None, flagbackup=None, flagsum=None, flagdetailedsum=None, parallel=None) → ResultsList[Results]

Apply precomputed calibrations to the data.

*hif\_applycal* applies the precomputed calibration tables stored in the pipeline context to the set of visibility files using predetermined field and spectral window maps and default values for the interpolation schemes.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets in the pipeline context.  
Example: `['X227.ms']`
- **field** -- A string containing the list of field names or field ids to which the calibration will be applied. Defaults to all fields in the pipeline context.  
Examples: `'3C279', '3C279, M82'`
- **intent** -- A string containing the list of intents against which the selected fields will be matched. Defaults to all supported intents in the pipeline context.  
Example: `'*TARGET*'`
- **spw** -- The list of spectral windows and channels to which the calibration will be applied. Defaults to all science windows in the pipeline context.  
Examples: `'17', '11, 15'`
- **antenna** -- The selection of antennas to which the calibration will be applied. Defaults to all antennas. Not currently supported.
- **parang** -- Apply parallactic angle correction
- **applymode** -- Calibration apply mode. Options:
  - `'calflag'`: calibrate data and apply flags from solutions
  - `'calflagstrict'`: (default) same as above except flag spws for which calibration is unavailable in one or more tables (instead of allowing them to pass uncalibrated and unflagged)
  - `'trial'`: report on flags from solutions, dataset entirely unchanged
  - `'flagonly'`: apply flags from solutions only, data not calibrated
  - `'flagonlystrict'`: same as above except flag spws for which calibration is unavailable in one or more tables
  - `'calonly'`: calibrate data only, flags from solutions **not** applied
- **calwt** -- Calibrate the weights as well as the data
- **flagbackup** -- Backup the flags before the apply
- **flagsum** -- Compute before and after flagging summary statistics

- **flagdetailedsum** -- Compute detailed before and after flagging statistics summaries. Parameter available only when if flagsum is True.
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: 'automatic', 'true', 'false', True, False  
Default: None (equivalent to False)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Apply the calibration to the target data

```
>>> hif_applycal(intent='TARGET')
```

## 2.3 pipeline.hif.cli.hif\_checkproductsize

**hif\_checkproductsize**(vis=None, maxcubesize=None, maxcubelimit=None, maxproductsize=None, maximsize=None, calcsb=None, parallel=None) → Results

Check imaging product size.

Check interferometry imaging product size and try to mitigate to maximum allowed values. *hif\_checkproductsize* implements a mitigation cascade computing the largest cube size and tries to reduce it below a given limit by adjusting the **nbins**, **hm\_imsize** and **hm\_cell** parameters. If this step succeeds, it also checks the overall imaging product size and if necessary reduces the number of fields to be imaged.

Alternatively, if **maximsize** is set, the image product pixel count is mitigated by trying to adjust **hm\_cell** parameter. If the pixel count is still greater than **maximsize** at **hm\_cell** of 4ppb, then this value is kept and the image field is truncated around the phase center by forcing **hm\_imsize** to **maximsize**.

Note that mitigation for image pixel count and for the product size currently are mutually exclusive, with **maximsize** taking precedence if set.

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in *hifa\_importdata* and *hifv\_importdata*.  
Examples: 'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']
- **maxcubesize** -- Maximum allowed cube size in gigabytes (mitigation goal)  
Default: -1 - automatic from performance parameters
- **maxcubelimit** -- Maximum allowed cube limit in gigabytes (mitigation failure limit)  
Default: -1 - automatic from performance parameters
- **maxproductsize** -- Maximum allowed product size in gigabytes (mitigation goal and failure limit)  
Default: -1 - automatic from performance parameters
- **maximsize** -- Maximum allowed image count size (mitigation goal and hard maximum). Parameter **maximsize** must be even and divisible by 2, 3, 5, 7 only. \*Caution\* **maximsize** is disabled by default and cannot be set at the same time as **maxcubesize**, **maxcubelimit** and **maxproductsize**.  
Default: -1 - disables mitigation for this parameter
- **calcsb** -- Force (re-)calculation of sensitivities and beams

- **parallel** -- Use the CASA imager parallelization when possible.  
Options: 'automatic', 'true', 'false', True, False  
Default: 'automatic'

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Basic call to check the product sizes using internal defaults

```
>>> hif_checkproductsize()
```

2. Typical ALMA call

```
>>> hif_checkproductsize(maxcubesize=40.0, maxcubelimit=60.0, maxproductsize=350.0)
```

## 2.4 pipeline.hif.cli.hif\_correctedampflag

**hif\_correctedampflag**(vis=None, intent=None, field=None, spw=None, antnegsig=None, antpossig=None, tmantint=None, tmint=None, tmbt=None, antblnegsig=None, antblpossig=None, relaxed\_factor=None, niter=None) → ResultsList[Results]

Flag corrected - model amplitudes based on calibrators.

**hif\_correctedampflag** looks for outlier visibility points by statistically examining the scalar difference of corrected amplitudes minus model amplitudes, and flags those outliers. The philosophy is that only outlier data points that have remained outliers after calibration will be flagged. The heuristic works equally well on resolved calibrators and point sources because it is not performing a vector difference, and thus is not sensitive to nulls in the flux density vs. uvdistance domain. Note that the phase of the data is not assessed.

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the *hifa\_importdata* and *hifv\_importdata*.  
Examples: 'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']
- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched. If undefined (default), it will select all data with the BANDPASS intent.  
Example: **intent='\*PHASE\*'**
- **field** -- The list of field names or field ids for which bandpasses are computed. If undefined (default), it will select all fields.  
Examples: '3C279', '3C279, M82'
- **spw** -- The list of spectral windows and channels for which bandpasses are computed. If undefined (default), it will select all science spectral windows.  
Example: **spw='11,13,15,17'**
- **antnegsig** -- Lower sigma threshold for identifying outliers as a result of bad antennas within individual timestamps
- **antpossig** -- Upper sigma threshold for identifying outliers as a result of bad antennas within individual timestamps

- **tmantint** -- Threshold for maximum fraction of timestamps that are allowed to contain outliers
- **tmint** -- Initial threshold for maximum fraction of "outlier timestamps" over "total timestamps" that a baseline may be a part of
- **tmb1** -- Initial threshold for maximum fraction of "bad baselines" over "all timestamps" that an antenna may be a part of
- **antblnegsig** -- Lower sigma threshold for identifying outliers as a result of "bad baselines" and/or "bad antennas" within baselines (across all timestamps)
- **antblpossig** -- Upper sigma threshold for identifying outliers as a result of "bad baselines" and/or "bad antennas" within baselines (across all timestamps)
- **relaxed\_factor** -- Relaxed value to set the threshold scaling factor to under certain conditions (see task description)
- **niter** -- Maximum number of times to iterate on evaluation of flagging heuristics. If an iteration results in no new flags, then subsequent iterations are skipped.

### Returns

The results object for the pipeline task is returned.

### Examples

Run default flagging on bandpass calibrator with recommended settings:

```
>>> hif_correctedampflag()
```

## 2.5 pipeline.hif.cli.hif\_editimlist

**hif\_editimlist**(*imagename=None*, *search\_radius\_arcsec=None*, *cell=None*, *cfcache=None*, *conjbeams=None*, *cyclefactor=None*, *cycleniter=None*, *nmajor=None*, *datatype=None*, *datacolumn=None*, *deconvolver=None*, *editmode=None*, *field=None*, *imaging\_mode=None*, *imsize=None*, *intent=None*, *gridder=None*, *mask=None*, *pbmask=None*, *nbin=None*, *nchan=None*, *niter=None*, *nterms=None*, *parameter\_file=None*, *pblimit=None*, *phasecenter=None*, *refreq=None*, *restfreq=None*, *robust=None*, *scales=None*, *specmode=None*, *spw=None*, *start=None*, *stokes=None*, *sensitivity=None*, *threshold=None*, *nsigma=None*, *uv taper=None*, *uvrange=None*, *width=None*, *vlass\_plane\_reject\_ms=None*) → Results

Add to a list of images to be produced with **hif\_makeimages**.

### Parameters

- **imagename** -- Prefix for output image names.
- **search\_radius\_arcsec** -- Size of the field finding beam search radius in arcsec.
- **cell** -- Image cell size(s) in X and Y, specified in angular units or pixels per beam.
  - A single value applies to both axes.
  - Use the format '**<number>ppb**' to specify pixels per beam.

By default, the cell size is computed from the UV coverage of all fields to be imaged, assuming a sampling of 5 pixels per beam, i.e., '5ppb'. When using the pixels-per-beam format (e.g., '3ppb'), the cell size is scaled accordingly.

Examples: ['0.5arcsec', '0.5arcsec'], '3ppb'

- **cfcache** -- Convolution function cache directory name
- **conjbeams** -- Use conjugate frequency in tclean for wideband A-terms.

- **cyclefactor** -- Controls the depth of clean in minor cycles based on PSF.
- **cycleniter** -- Controls max number of minor cycle iterations in a single major cycle.
- **nmajor** -- Controls the maximum number of major cycles to evaluate.
- **datatype** -- Data type(s) to image. The default '' selects the best available data type (e.g. selfcal over regcal) with an automatic fallback to the next available data type. With the **datatype** parameter of '**regcal**' or '**selfcal**', one can force the use of only given data type(s). Note that this parameter is only for non-VLASS data when the datacolumn is not explicitly set by user or imaging heuristics.
- **datacolumn** -- Data column to image; this will take precedence over the datatype parameter.
- **deconvolver** -- Minor cycle algorithm (multiscale or mtmfs)
- **editmode** -- The edit mode of the task ('**add**' or '**replace**'). Defaults to '**add**'.
- **field** -- Set of data selection field names or ids.
- **imaging\_mode** -- Identity of product type (e.g. VLASS quick look) desired. This will determine the heuristics used.
- **imsize** -- Image X and Y size(s) in pixels or PB level (single fields), '' for default. Single value same for both. '**<number>pb**' for PB level.
- **intent** -- Set of data selection intents
- **gridder** -- Name of the gridder to use with tclean
- **mask** -- Used to declare whether to use a predefined mask for tclean.
- **pbmask** -- Used to declare primary beam gain level for cleaning with primary beam mask (**usemask='pb'**), used only for VLASS-SE-CONT imaging mode.
- **nbin** -- Channel binning factor.
- **nchan** -- Number of channels,  
Default: **-1**, which means all channels.
- **niter** -- The max total number of minor cycle iterations allowed for tclean
- **nterms** -- Number of Taylor coefficients in the spectral model
- **parameter\_file** -- keyword=value text file as alternative method of input parameters
- **pblimit** -- PB gain level at which to cut off normalizations
- **phassecenter** -- The default phase center is set to the mean of the field directions of all fields that are to be image together.  
Example: **0, 'J2000 19h30m00 -40d00m00'**
- **reffreq** -- Reference frequency of the output image coordinate system
- **restfreq** -- List of rest frequencies or a rest frequency in a string for output image.
- **robust** -- Briggs robustness parameter for tclean
- **scales** -- The scales for multi-scale imaging.
- **specmode** -- Spectral gridding type. Options: '**mfs**', '**cont**', '**cube**', ''.
- **spw** -- Set of data selection spectral window/channels, '' for all
- **start** -- First channel for frequency mode images. Starts at first input channel of the spw.  
Example: **'22.3GHz'**

- **stokes** -- Stokes Planes to make
- **sensitivity** -- Theoretical sensitivity (override internal calculation)
- **threshold** -- Stopping threshold (number in units of Jy, or string)
- **nsigma** -- Multiplicative factor for rms-based threshold stopping
- **uv taper** -- Used to set a uv-taper during clean.
- **uvrange** -- Set of data selection uv ranges, '' for all.
- **width** -- Channel width
- **vlass\_plane\_reject\_ms** (*bool or dict, optional*) -- Control VLASS Coarse Cube plane rejection based on flagging percentages. Only applies to the '**'VCLASS-SE-CUBE'** imaging mode.

Default is **True**, which automatically rejects planes with high flagging percentages using built-in heuristics (see details below).

Options:

- **True**: Enable automatic plane rejection with default thresholds.
- **False**: Disable flagging-based plane rejection entirely.
- **dict**: Enable plane rejection with custom threshold parameters.

When providing a dictionary, supported keys are:

- **exclude\_spw** (str, default ''): Comma-separated list of spectral windows to exclude from rejection consideration (always preserved).
- **flagpct\_thresh** (float, default **0.9**): Flagging percentage threshold per field for triggering plane rejection.
- **nfield\_thresh** (int, default **12**): Minimum number of fields that must exceed the flagging threshold before rejecting the plane.

#### Returns

The results object for the pipeline task is returned.

## 2.6 pipeline.hif.cli.hif\_findcont

**hif\_findcont**(*vis=None, target\_list=None, hm\_mosweight=None, hm\_perchanweightdensity=None, hm\_weighting=None, datacolumn=None, parallel=None*) → Results

Find continuum frequency ranges for a list of specified targets.

If a *cont.dat* file is not already present in the working directory, then dirty image cubes are created for each spectral window of each science target at the native channel resolution unless the **nbins** parameter was used in the preceding *hif\_makeimlist* stage. Robust=1 Briggs weighting is used for optimal line sensitivity, even if a different robust had been chosen in *hifa\_imageprecheck* to match the PI requested angular resolution. Using moment0 and moment8 images of each cube, SNR-based masks are created, and the mean spectrum of the joint mask is computed and evaluated with extensive heuristics to find the channel ranges that are likely to be free of line emission. Warnings are generated if the channel ranges contain a small fraction of the bandwidth, or sample only a limited extent of the spectrum.

If a *cont.dat* file already exists in the working directory before this task is executed, then it will first examine the contents. For any spw that already has frequency ranges defined in this file, it will not perform the analysis described above in favor of the a priori ranges. For spws not listed in a pre-existing file, it will analyze them as normal and update the file. In either case, the *cont.dat* file is used by the subsequent *hif\_uvcontsub* and *hif\_makeimages* stages.

## Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the <hifa,hifv>\_importdata task. ": use all MeasurementSets in the context  
Examples: 'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']
- **target\_list** -- Dictionary specifying targets to be imaged; blank will read list from context.
- **hm\_mosweight** -- Mosaic weighting. Defaults to " to enable the automatic heuristics calculation. Can be set to True or False manually.
- **hm\_perchanweightdensity** -- Calculate the weight density for each channel independently. Defaults to " to enable the automatic heuristics calculation. Can be set to True or False manually.
- **hm\_weighting** -- Weighting scheme (natural,uniform,briggs,briggsabs[experimental],briggsbwtpaper[experimental])
- **datacolumn** -- Data column to image. Only to be used for manual overriding when the automatic choice by data type is not appropriate.
- **parallel** -- Use CASA/tclean built-in parallel imaging when possible.  
Options: 'automatic', 'true', 'false', True, False  
Default: 'automatic'

## Returns

The results object for the pipeline task is returned.

## Examples

1. Perform continuum frequency range detection for all science targets and spws:

```
>>> hif_findcont()
```

## 2.7 pipeline.hif.cli.hif\_lowgainflag

**hif\_lowgainflag**(vis=None, intent=None, spw=None, refant=None, flag\_nmedian=None, fnm\_lo\_limit=None, fnm\_hi\_limit=None, tmeff\_limit=None) → ResultsList[Results]

Flag antennas with low or high gain.

Deviant antennas are detected by outlier analysis of a view showing their amplitude gains, pre-applying a temporary bandpass and phase solution. This view is a list of 2D images with axes 'Scan' and 'Antenna'; there is one image for each spectral window and intent. A flagcmd to flag all data for an antenna will be generated by any gain that is outside the range [fnm\_lo\_limit \* median, fnm\_hi\_limit \* median].

## Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the <hifa,hifv>\_importdata task. ": use all MeasurementSets in the context  
Examples: 'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']
- **intent** -- A string containing the list of intents to be checked for antennas with deviant gains. The default is blank, which causes the task to select the 'BANDPASS' intent.
- **spw** -- The list of spectral windows and channels to which the calibration will be applied. Defaults to all science windows in the pipeline context.  
Examples: spw='17', spw='11, 15'

- **refant** -- A string containing a prioritized list of reference antenna name(s) to be used to produce the gain table. Defaults to the value(s) stored in the pipeline context. If undefined in the pipeline context defaults to the CASA reference antenna naming scheme.

Examples: refant='DV01', refant='DV06,DV07'

- **flag\_nmedian** -- Whether to flag figures of merit greater than **fmm\_hi\_limit** \* median or lower than **fmm\_lo\_limit** \* median. (default: True)
- **fmm\_lo\_limit** -- Flag values lower than **fmm\_lo\_limit** \* median (default: 0.5)
- **fmm\_hi\_limit** -- Flag values higher than **fmm\_hi\_limit** \* median (default: 1.5)
- **tmeff1\_limit** -- Threshold for "too many entirely flagged" - the critical fraction of antennas whose solutions are entirely flagged in the flagging view of a spw for this stage: if the fraction is equal or greater than this value, then flag the visibility data from all antennas in this spw (default: 0.666)

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Flag antennas with low or high gain using recommended thresholds:

```
>>> hif_lowgainflag()
```

## 2.8 pipeline.hif.cli.hif\_makecutoutimages

**hif\_makecutoutimages**(vis=None, offsetblc=None, offsettrc=None) → Results

Cutout central 1 sq. degree from VLASS QL, SE, and Coarse Cube images.

#### Parameters

- **vis** -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSs, or tar files of MSs. If ASDM files are specified, they will be converted to MS format. example: **vis=['X227.ms', 'asdms.tar.gz']**
- **offsetblc** -- -x and -y offsets to the bottom lower corner (blc) in arcseconds
- **offsettrc** -- +x and +y offsets to the top right corner (trc) in arcseconds

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Basic makecutoutimages task

```
>>> hif_makecutoutimages()
```

## 2.9 pipeline.hif.cli.hif\_makeimages

**hif\_makeimages**(vis=None, target\_list=None, hm\_sidelobethreshold=None, hm\_noisethreshold=None, hm\_lownoisethreshold=None, hm\_negativethreshold=None, hm\_minbeamfrac=None, hm\_growiterations=None, hm\_dogrowprune=None, hm\_minpercentchange=None, hm\_fastnoise=None, hm\_nsigma=None, hm\_perchanweightdensity=None, hm\_npixels=None, hm\_cyclefactor=None, hm\_nmajor=None, hm\_minpsffraction=None, hm\_maxpsffraction=None, hm\_weighting=None, hm\_cleaning=None, tlimit=None, drcorrect=None, masklimit=None, cleancontranges=None, calcsb=None, hm\_mosweight=None, overwrite\_on\_export=None, vlass\_plane\_reject\_im=None, parallel=None) → Results

Compute clean results from a list of specified targets.

### Parameters

- **`vis`** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the <hifa,hifv>\_importdata task. ":" use all MeasurementSets in the context Examples: 'ngc5921.ms', ['ngc5921a.ms', ngc5921b.ms', 'ngc5921c.ms']
- **`target_list`** -- Dictionary specifying targets to be imaged; blank will read list from context
- **`hm_masking`** -- Clean masking mode. Options are 'centralregion', 'auto', 'manual' and 'none'
- **`hm_sidelobethreshold`** -- sidelobethreshold \* the max sidelobe level
- **`hm_noisethreshold`** -- noisethreshold \* rms in residual image
- **`hm_lownoisethreshold`** -- lownoisethreshold \* rms in residual image
- **`hm_negativethreshold`** -- negativethreshold \* rms in residual image
- **`hm_minbeamfrac`** -- Minimum beam fraction for pruning
- **`hm_growiterations`** -- Number of binary dilation iterations for growing the mask
- **`hm_dogrowprune`** -- Do pruning on the grow mask. Defaults to " to enable the automatic heuristics calculation. Can be set to True or False manually.
- **`hm_minpercentchange`** -- Mask size change threshold
- **`hm_fastnoise`** -- Faster noise calculation for automask or nsigma stopping. Defaults to " to enable the automatic heuristics calculation. Can be set to True or False manually.
- **`hm_nsigma`** -- Multiplicative factor for rms-based threshold stopping
- **`hm_perchanweightdensity`** -- Calculate the weight density for each channel independently. Defaults to " to enable the automatic heuristics calculation. Can be set to True or False manually.
- **`hm_npixels`** -- Number of pixels to determine uv-cell size for super-uniform weighting
- **`hm_cyclefactor`** -- Scaling on PSF sidelobe level to compute the minor-cycle stopping threshold
- **`hm_nmajor`** -- Controls the maximum number of major cycles to evaluate.
- **`hm_minpsffraction`** -- PSF fraction that marks the max depth of cleaning in the minor cycle
- **`hm_maxpsffraction`** -- PSF fraction that marks the minimum depth of cleaning in the minor cycle
- **`hm_weighting`** -- Weighting scheme (natural,uniform,briggs,briggsabs[experimental],briggsbwtpaper[experimental])
- **`hm_cleaning`** -- Pipeline cleaning mode
- **`tlimit`** -- Times the sensitivity limit for cleaning
- **`drcorrect`** -- Override the default heuristics-based DR correction (for ALMA data only)
- **`masklimit`** -- Times good mask pixels for cleaning
- **`cleancontranges`** -- Clean continuum frequency ranges in cubes
- **`calcsb`** -- Force (re-)calculation of sensitivities and beams
- **`hm_mosweight`** -- Mosaic weighting Defaults to " to enable the automatic heuristics calculation. Can be set to True or False manually.
- **`overwrite_on_export`** -- Replace existing image products when h/hifa/hifv\_exportdata is called. If False, images that would have the same FITS name on export, are amended to include a version number. For example, if oussid.J1248-4559\_ph.spw21.mfs.I.pbcor.fits would already be exported by a previous call to

hif\_makeimags, then 'oussid.J1248-4559\_ph.spw21.mfs.I.pbcor.v2.fits' would also be exported to the products/ directory. The first exported product retains the same name. Additional products start counting with 'v2', 'v3', etc.

- **vlass\_plane\_reject\_im** -- Only used for the 'VLASS-SE-CUBE' imaging mode.

Default: True

If True, reject VLASS Coarse Cube planes with high flagging percentages or outlier beam sizes (see the heuristics details below)

If False, do not perform the post-imaging VLASS Coarse Cube plane rejection. If the input value is a dictionary, the plane rejection heuristics will be performed with custom thresholds. The optional keys could be:

- exclude\_spw, default: " Spectral windows to be excluded from the VLASS Coarse Cube post-imaging plane rejection consideration, i.e. always preserve.
- flagpct\_thresh, default: 0.8 The flagging percentage across the entire mosaic to be considered to be high flagging level for the plane rejection.
- beamdev\_thresh: default: 0.2 Threshold for the fractional beam deviation from the expected value required for the plane rejection.

- **parallel** -- Use CASA/tclean built-in parallel imaging for individual scientific targets, or, perform continuum imaging of multiple target (calibrators) concurrently without the CASA/tclean built-in parallelization.

Options: '**automatic**', '**true**', '**false**', **True**, **False**

**Default: '**automatic**'** - optimizes the parallelization mode based on the imaging target type (scientific targets vs. calibrators) and the specific operation.

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Compute clean results for all imaging targets defined in a previous hif\_makeimlist or hif\_editimlist call:

```
>>> hif_makeimages()
```

2. Compute clean results overriding automatic masking choice:

```
>>> hif_makeimages(hm_masking='centralregion')
```

## 2.10 pipeline.hif.cli.hif\_makeimlist

**hif\_makeimlist**(vis=None, imagename=None, intent=None, field=None, spw=None, stokes=None, conffile=None, linesfile=None, uvrage=None, specmode=None, outframe=None, hm\_imsize=None, hm\_cell=None, calmaxpix=None, phasecenter=None, nchan=None, start=None, width=None, nbins=None, robust=None, uvtaper=None, clearlist=None, per\_eb=None, per\_session=None, calcbs=None, datatype=None, datacolumn=None, allow\_wproject=None, parallel=None) → Results

Compute list of clean images to be produced.

Generate a list of images to be cleaned. By default, the list will include one image per science target per spw. Calibrator targets can be selected by setting appropriate values for **intent**.

By default, the output image cell size is set to the minimum cell size consistent with the UV coverage.

By default, the image size in pixels is set to values determined by the cell size and the primary beam size. If a calibrator is being imaged (intents 'PHASE', 'BANDPASS', 'FLUX' or 'AMPLITUDE') then the image dimensions are limited to 'calmaxpix' pixels.

By default, science target images are cubes and calibrator target images are mfs. Science target images may be mosaics or single fields.

## Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the <hifa,hifv>\_importdata task. "": use all MeasurementSets in the context

Examples: 'ngc5921.ms', ['ngc5921a.ms', ngc5921b.ms', 'ngc5921c.ms']

- **imagename** -- Prefix for output image names, "" for automatic.

- **intent** -- Select intents for which associated fields will be imaged. Possible choices are PHASE, BANDPASS, AMPLITUDE, CHECK and TARGET or combinations thereof.

Examples: 'PHASE,BANDPASS', 'TARGET'

- **field** -- Select fields to image. Use field name(s) NOT id(s). Mosaics are assumed to have common source / field names. If intent is specified only fields with data matching the intent will be selected. The fields will be selected from MeasurementSets in "vis". "": Fields matching intent, one image per target source.

- **spw** -- Select spectral windows to image. "": Images will be computed for all science spectral windows.

- **stokes** -- Select the Stokes parameters to image. "": Stokes I will be computed except for polarization calibrators, where the automatic heuristics selects IQUV. Setting a value here will override the heuristics. Allowed values are 'T' and 'IQUV'.

- **contfile** -- Name of file with frequency ranges to use for continuum images.

- **linesfile** -- Name of file with line frequency ranges to exclude for continuum images.

- **uvrange** -- Select a set of uv ranges to image. "": All uv data is included

Examples: '0~1000klambda', ['0~100klambda', 100~1000klambda]

- **specmode** -- Frequency imaging mode, 'mfs', 'cont', 'cube', 'repBW'. " defaults to 'cube' if **intent** parameter includes 'TARGET' otherwise 'mfs'. specmode='mfs' produce one image per source and spw specmode='cont' produce one image per source and aggregate over all specified spws specmode='cube' produce an LSRK frequency cube, channels are specified in frequency specmode='repBW' produce an LSRK frequency cube at representative channel width

- **outframe** -- velocity frame of output image (LSRK, " for automatic) (not implemented)

- **hm\_imsize** -- Image X and Y size in pixels or PB level for single fields. The explicit sizes must be even and divisible by 2,3,5,7 only. The default values are derived as follows: 1. Determine phase center and spread of field centers around it. 2. Set the size of the image to cover the spread of field centers plus a border of width 0.75 \* beam radius, to first null. 3. Divide X and Y extents by cell size to arrive at the number of pixels required. The PB level setting for single fields leads to an imsize extending to the specified level plus 5% padding in all directions.

Examples: '0.3pb', [120, 120]

- **hm\_cell** -- Image X and Y cell sizes. "": computes the cell size based on the UV coverage of all the fields to be imaged and uses a 5 pix per beam sampling. The pix per beam specification ('<number>ppb') uses the above default cell size ('5ppb') and scales it accordingly. The cells can also be specified as explicit measures.

Examples: '3ppb', ['0.5arcsec', '0.5arcsec']

- **calmaxpix** -- Maximum image X or Y size in pixels if a calibrator is being imaged ('PHASE', 'BANDPASS', 'AMPLITUDE' or 'FLUX' intent).

- **phasecenter** -- Direction measure or field id of the image center. The default phase center is set to the mean of the field directions of all fields that are to be imaged together.

Examples: 'J2000 19h30m00 -40d00m00', 0

- **nchan** -- Total number of channels in the output image(s) -1 selects enough channels to cover the data selected by spw consistent with start and width.

- **start** -- Start of image frequency axis as frequency or velocity. "" selects start frequency automatically.

- **width** -- Output channel width. Difference in frequency between 2 selected channels for frequency mode images. 'pilotimage' for 15 MHz / 8 channel heuristic

- **nbins** -- Channel binning factors for each spw. Format: 'spw1:nb1,spw2:nb2,...' with optional wildcards: '\*:nb'

Examples: '9:2,11:4,13:2,15:8', '\*:2'

- **robust** -- Briggs robustness parameter Values range from -2.0 (uniform) to 2.0 (natural)

- **uv taper** -- uv-taper on outer baselines

- **clearlist** -- Clear any existing target list

- **per\_eb** -- Make an image target per EB

- **per\_session** -- Make an image target per session

- **calcsb** -- Force (re-)calculation of sensitivities and beams

- **datatype** -- Data type(s) to image. The default " selects the best available data type (e.g. selfcal over regcal) with an automatic fallback to the next available data type. With the **datatype** parameter one can force the use of only given data type(s) without a fallback. The data type(s) are specified as comma separated string of keywords. Accepted values are the standard data types such as 'REGCAL\_CONTLINE\_ALL', 'REGCAL\_CONTLINE\_SCIENCE', 'SELF CAL\_CONTLINE\_SCIENCE', 'REGCAL\_LINE\_SCIENCE', 'SELF CAL\_LINE\_SCIENCE'. The shortcuts 'regcal' and 'selfcal' are also accepted. They are expanded into the full data types using the **specmode** parameter and the available data types for the given MSes. In addition the strings 'best' and 'all' are accepted, where 'best' means the above mentioned automatic mode and 'all' means all available data types for a given specmode. The data type strings are case insensitive.

Examples: 'selfcal', 'regcal', 'selfcal,regcal', 'REGCAL\_LINE\_SCIENCE,selfcal\_line\_science'

- **datacolumn** -- Data column to image. Only to be used for manual overriding when the automatic choice by data type is not appropriate.

- **allow\_wproject** -- Allow the wproject heuristics for imaging

- **parallel** -- Use the CASA imager parallel processing when possible.

Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`

Default: `'automatic'`

## Returns

The results object for the pipeline task is returned.

## Examples

1. Make a list of science target images to be cleaned, one image per science spw.

```
>>> hif_makeimlist()
```

2. Make a list of PHASE and BANDPASS calibrator targets to be imaged, one image per science spw.

```
>>> hif_makeimlist(intent='PHASE,BANDPASS')
```

3. Make a list of PHASE calibrator images observed in spw 1, images limited to 50 pixels on a side.

```
>>> hif_makeimlist(intent='PHASE', spw='1', calmaxpix=50)
```

## 2.11 pipeline.hif.cli.hif\_makermssimages

**hif\_makermssimages**(vis=None) → Results

Create RMS images for VLASS data.

### Parameters

**vis** -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSs, or tar files of MSs, If ASDM files are specified, they will be converted to MS format.

Example: `vis=['X227.ms', 'asdms.tar.gz']`

### Returns

The results object for the pipeline task is returned.

### Examples

1. Basic makermssimages task

```
>>> hif_makermssimages()
```

## 2.12 pipeline.hif.cli.hif\_mstransform

**hif\_mstransform**(vis=None, outputvis=None, field=None, intent=None, spw=None, chanbin=None, timebin=None, parallel=None) → ResultsList[Results]

Create new MeasurementSets for science target imaging.

Create new MeasurementSets for imaging from the corrected column of the input MeasurementSet via a single call to mstransform with all data selection parameters. By default, all science target data is copied to the new MS. The new MeasurementSet is not re-indexed to the selected data and the new MS will have the same source, field, and spw names and ids as it does in the parent MS.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the <hifa,hifv>\_importdata task. ":" use all MeasurementSets in the context

Examples: `'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']`

- **outputvis** -- A list of output MeasurementSets for line detection and imaging,. This list must have the same length as the input list.

Default Naming: By default, an input MS named <msrootname>.ms will produce an output named <msroot-name>\_targets.ms.

### Examples

- `outputvis='ngc5921_targets.ms'`

- `outputvis=['ngc5921a_targets.ms', 'ngc5921b_targets.ms', 'ngc5921c_targets.ms']`

- **field** -- Select fields name(s) or id(s) to transform. Only fields with data matching the intent will be selected.  
Examples: '3C279', 'Centaurus\*', '3C279,J1427-421'
- **intent** -- Select intents for which associated fields will be imaged. By default only TARGET data is selected.  
Examples: 'PHASE,BANDPASS'
- **spw** -- Select spectral window/channels to image. By default all science spws for which the specified intent is valid are selected.
- **chanbin** -- Width (bin) of input channels to average to form an output channel. If chanbin > 1 then chanaverage is automatically switched to True.
- **timebin** -- Bin width for time averaging. If timebin > 0s then timeaverage is automatically switched to True.
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: 'automatic', 'true', 'false', True, False  
Default: None (equivalent to False)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Create a science target MS from the corrected column in the input MS.

```
>>> hif_mstransform()
```

2. Make a phase and bandpass calibrator targets MS from the corrected column in the input MS.

```
>>> hif_mstransform(intent='PHASE,BANDPASS')
```

## 2.13 pipeline.hif.cli.hif\_rawflagchans

**hif\_rawflagchans**(vis=None, spw=None, intent=None, flag\_hilo=None, fhl\_limit=None, fhl\_minsample=None, flag\_bad\_quadrant=None, fbq\_hilo\_limit=None, fbq\_antenna\_frac\_limit=None, fbq\_baseline\_frac\_limit=None, parallel=None) → ResultsList[Results]

Flag deviant baseline/channels in raw data.

hif\_rawflagchans flags deviant baseline/channels in the raw data.

The flagging views used are derived from the raw data for the specified intent - default is BANDPASS.

Bad baseline/channels are flagged for all intents, not just the one that is the basis of the flagging views.

For each spectral window the flagging view is a 2d image with axes 'channel' and 'baseline'. The pixel for each channel, baseline is the time average of the underlying unflagged raw data.

The baseline axis is labeled by numbers of form id1.id2 where id1 and id2 are the IDs of the baseline antennas. Both id1 and id2 run over all antenna IDs in the observation. This means that each baseline is shown twice but has the benefit that 'bad' antennas are easily identified by eye.

Three flagging methods are available:

If parameter **flag\_hilo** is set True then outliers from the median of each flagging view will be flagged.

If parameter **flag\_bad\_quadrant** is set True then a simple 2 part test is used to check for bad antenna quadrants and/or bad baseline quadrants. Here a 'quadrant' is defined simply as one quarter of the channel axis. The first part of the test is to note as 'suspect'

those points further from the view median than `fbq_hilo_limit` \* MAD. The second part is to flag entire antenna/quadrants if their fraction of suspect points exceeds `fbq_antenna_frac_limit`. Failing that, entire baseline/quadrants may be flagged if their fraction of suspect points exceeds `fbq_baseline_frac_limit`. Suspect points are not flagged unless as part of a bad antenna or baseline quadrant.

### Parameters

- `vis` -- List of input MeasurementSets. default: [] - Use the MeasurementSets currently known to the pipeline context.
- `spw` -- The list of spectral windows and channels to which the calibration will be applied. Defaults to all science windows in the pipeline context.  
Example: `spw='17', spw='11, 15'`
- `intent` -- A string containing the list of intents to be checked for antennas with deviant gains. The default is blank, which causes the task to select the 'BANDPASS' intent.  
Example: `intent='*BANDPASS*'`
- `flag_hilo` -- True to flag channel/baseline data further from the view median than `fhl_limit` \* MAD.
- `fhl_limit` -- If `flag_hilo` is True then flag channel/baseline data further from the view median than `fhl_limit` \* MAD.
- `fhl_minsample` -- Do no flagging if the view median and MAD are derived from fewer than `fhl_minsample` view pixels.
- `flag_bad_quadrant` -- True to search for and flag bad antenna quadrants and baseline quadrants. Here a '/quadrant/' is one quarter of the channel axis.
- `fbq_hilo_limit` -- If `flag_bad_quadrant` is True then channel/baselines further from the view median than `fbq_hilo_limit` \* MAD will be noted as 'suspect'. If there are enough of them to indicate that an antenna or baseline quadrant is bad then all channel/baselines in that quadrant will be flagged.
- `fbq_antenna_frac_limit` -- If `flag_bad_quadrant` is True and the fraction of suspect channel/baselines in a particular antenna/quadrant exceeds `fbq_antenna_frac_limit` then all data for that antenna/quadrant will be flagged.
- `fbq_baseline_frac_limit` -- If `flag_bad_quadrant` is True and the fraction of suspect channel/baselines in a particular baseline/quadrant exceeds `fbq_baseline_frac_limit` then all data for that baseline/quadrant will be flagged.
- `parallel` -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: `'automatic', 'true', 'false', True, False`  
Default: `None` (equivalent to `False`)

### Returns

The results object for the pipeline task is returned.

### Examples

1. Flag bad quadrants and wild outliers, default method:

```
>>> hif_rawflagchans()
```

equivalent to:

```
>>> hif_rawflagchans(flag_hilo=True, fhl_limit=20, flag_bad_quadrant=True, fbq_hilo_limit=8,
...                      fbq_antenna_frac_limit=0.2, fbq_baseline_frac_limit=1.0)
```

## 2.14 pipeline.hif.cli.hif\_refant

**hif\_refant**(vis=None, field=None, spw=None, intent=None, hm\_refant=None, refant=None, geometry=None, flagging=None, parallel=None, refantignore=None) → ResultsList[Results]

Select the best reference antennas.

The hif\_refant task selects a list of reference antennas and stores them in the pipeline context in priority order.

The priority order is determined by a weighted combination of scores derived by the antenna selection heuristics. In manual mode the reference antennas can be set by hand.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets in the pipeline context.  
Example: ['M31.ms']
- **field** -- The comma delimited list of field names or field ids for which flagging scores are computed if hm\_refant='automatic' and flagging = True  
Example: " (Default to fields with the specified intents), '3C279', '3C279,M82'
- **spw** -- A string containing the comma delimited list of spectral window ids for which flagging scores are computed if hm\_refant='automatic' and flagging = True.  
Example: " (all spws observed with the specified intents), '11,13,15,17'
- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched. Defaults to all supported intents.  
Example: 'BANDPASS', 'AMPLITUDE,BANDPASS,PHASE,POLARIZATION'
- **hm\_refant** -- The heuristics method or mode for selection the reference antenna. The options are 'manual' and 'automatic'. In manual mode a user supplied reference antenna refant is supplied. In 'automatic' mode the antennas are selected automatically.
- **refant** -- The user supplied reference antenna for hm\_refant='manual'. If no antenna list is supplied an empty list is returned.  
Example: 'DV05'
- **geometry** -- Score antenna by proximity to the center of the array. This option is quick as only the ANTENNA table must be read. Parameter is available when ``hm\_refant``='automatic'.
- **flagging** -- Score antennas by percentage of unflagged data. This option requires computing flagging statistics. Parameter is available when ``hm\_refant``='automatic'.
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: 'automatic', 'true', 'false', True, False  
Default: None (equivalent to False)
- **refantignore** -- string list to be ignored as reference antennas.  
Example: refantignore='ea02,ea03'

### Returns

The results object for the pipeline task is returned.

## Examples

- Compute the references antennas to be used for bandpass and gain calibration.

```
>>> hif_refant()
```

## 2.15 pipeline.hif.cli.hif\_selfcal

**hif\_selfcal**(vis=None, field=None, spw=None, contfile=None, hm\_imsize=None, hm\_cell=None, apply=None, recal=None, restore\_only=None, overwrite=None, refantignore=None, restore\_resources=None, n\_solints=None, amplitude\_selfcal=None, gaincal\_minsnr=None, minsnr\_to\_proceed=None, delta\_beam\_thresh=None, apply\_cal\_mode\_default=None, rel\_thresh\_scaling=None, dividing\_factor=None, check\_all\_spws=None, inf\_EB\_gaincal\_combine=None, usermask=None, usermodel=None, allow\_wproject=None, parallel=None) → Results

Determine and apply self-calibration with the science target data.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets defined in the pipeline context.
- **field**-- Select field(s) for self-calibration. Use field name(s) NOT id(s). Mosaics are assumed to have common source/field names.
- **spw**-- Select spectral windows for self-calibration.
- **contfile**-- Name of file to specify line-free frequency ranges for selfcal continuum imaging. Defaults to 'cont.dat'.
- **hm\_imsize**-- self-calibration imaging dimension in pixels, or PB level for single fields.
- **hm\_cell**-- self-calibration imaging cell size.
- **apply**-- Apply final selfcal solutions back to the input MeasurementSets. Defaults to **True**.
- **recal**-- Always re-do self-calibration even solutions/caltables are found in the Pipeline context or json restore file. Defaults to **False**.

Note that the selfcal solutions might not be applied if self-calibrated data labeled by the pipeline Datatypes already exists. See **overwrite** below.

- **restore\_only**-- Only attempt to apply pre-existing selfcal calibration tables and would not run the self-calibration sequence if their records (.selfcal.json, gaintables) are not present. Defaults to **False**. **restore\_only** will take precedence over **recal**.
- **overwrite**-- Allow overwriting pre-existing self-calibrated data of applicable field/spw labeled by DataType. Defaults to **False**.
- **refantignore**-- Antennas to be ignored as reference antennas. Defaults to '''. Examples:
  - **refantignore='ea02,ea03'**
  - **refantignore={'ms1.ms':'ea02,ea03','ms2.ms': 'ea03'}**, specified at the per-ms level.
- **restore\_resources**-- Path to the restore resources from a standard run of hif\_selfcal. hif\_selfcal will automatically do an exhaustive search to lookup/extract/verify the selfcal restore resources, i.e., selfcal.json and all selfcal-caltable referred in selfcal.json, starting from *working/*, to *products/* and *rawdata/*. If **restore\_resources** is specified, this file path will be evaluated first before the pre-defined exhaustive search list. The value can be the file path of *\*auxproducts.tgz* file or *\*selfcal.json* file.
- **n\_solints**-- number of solution intervals to attempt for self-calibration. Defaults to **4**.

- **amplitude\_selfcal** -- Attempt amplitude self-calibration following phase-only self-calibration; if median time between scans of a given target is < 150s, solution intervals of **300s** and **inf** will be attempted, otherwise just **inf** will be attempted. Defaults to **False**.
- **gaincal\_minsnr** -- Minimum S/N for a solution to not be flagged by gaincal. Defaults to **2.0**.
- **minsnr\_to\_proceed** -- Minimum estimated S/N on a per antenna basis to attempt self-calibration of a source. Defaults to **3.0**.
- **delta\_beam\_thresh** -- Allowed fractional change in beam size for selfcalibration to accept results of a solution interval. Defaults to **0.05**.
- **apply\_cal\_mode\_default** -- Apply mode to use for applycal task during self-calibration; same options as applycal. Defaults to '**calflag**'.
- **rel\_thresh\_scaling** -- Scaling type to determine how clean thresholds per solution interval should be determined going from the starting clean threshold to  $3.0 * \text{RMS}$  for the final solution interval. Defaults to '**log10**'. Available options: '**linear**', '**log10**', or '**loge**' (natural log)
- **dividing\_factor** -- Scaling factor to determine clean threshold for first self-calibration solution interval. Equivalent to  $(\text{Peak S/N} / \text{dividing\_factor}) * \text{RMS}$  as the first clean threshold; however, if  $(\text{Peak S/N} / \text{dividing\_factor}) < 5.0$ ; a value of 5.0 is used for the first clean threshold. Defaults to **40** for < 8 GHz or **15** for > 8 GHz.
- **check\_all\_spws** -- If **True**, the S/N of mfs images created on a per-spectral-window basis will be compared at the initial stages final self-calibration. Defaults to **False**.
- **inf\_EB\_gaincal\_combine** -- change gain solution combination parameters for the inf\_EB solution interval. if **True**, the gaincal combine parameter will be set to '**scan,spw**'; if **False**, the gaincal combine parameter will be set to '**scan**'. Defaults to **False**.
- **usermask** -- User mask to be used for self-calibration imaging. (not implemented)
- **usermodel** -- User model to be used for self-calibration imaging. (not implemented)
- **allow\_wproject** -- Allow the wproject heuristics for self-calibration imaging. Defaults to **False**.
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework, and use CASA/tclean parallel imaging, when possible.  
Options: '**automatic**', '**true**', '**false**', **True**, **False**  
Default: **None** (equivalent to '**automatic**')

### Returns

The results object for the pipeline task is returned.

### Examples

1. Run self-calibration and apply solutions to all science targets and spws

```
>>> hif_selfcal()
```

2. Run self-calibration and apply solutions to a single science target

```
>>> hif_selfcal(field="3C279")
```

3. Run self-calibration with a more relaxed allowed fractional change in the beam size for a solution interval to be successful

```
>>> hif_selfcal(delta_beam_thresh=0.15)
```

## 2.16 pipeline.hif.cli.hif\_setjy

**hif\_setjy**(vis=None, field=None, intent=None, spw=None, model=None, reffile=None, normfluxes=None, reffreq=None, fluxdensity=None, spix=None, scalebychan=None, standard=None) → ResultsList[Results]

Fill the model column with calibrated visibilities.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets defined in the pipeline context.

- **field** -- The list of field names or field ids for which the models are to be set. Defaults to all fields with intent '**\*AMPLITUDE\***'.

Example: `field='3C279', field='3C279, M82'`

- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched. Defaults to all data with amplitude intent.

Example: `intent='*AMPLITUDE*'`

- **spw** -- The list of spectral windows and channels for which bandpasses are computed. Defaults to all science spectral windows.

Example: `spw='11,13,15,17'`

- **model** -- Model image for setting model visibilities. Not fully supported.

Example: see details in help for CASA setjy task

- **reffile** -- Path to a file containing flux densities for calibrators unknown to CASA. Values given in this file take precedence over the CASA-derived values for all calibrators except solar system calibrators. By default the path is set to the CSV file created by `h_importdata`, consisting of catalogue fluxes extracted from the ASDM. example: `reffile='', reffile='working/flux.csv'`

- **normfluxes** -- Normalize lookup fluxes.

- **reffreq** -- The reference frequency for spix, given with units. Provided to avoid division by zero. If the flux density is being scaled by spectral index, then reffreq must be set to whatever reference frequency is correct for the given fluxdensity and spix. It cannot be determined from vis. On the other hand, if spix is 0, then any positive frequency can be used and will be ignored.

Example: `reffreq='86.0GHz', reffreq='4.65e9Hz'`

- **fluxdensity** -- Specified flux density [I,Q,U,V] in Jy. Uses [1,0,0,0] flux density for unrecognized sources, and standard flux densities for ones recognized by 'standard', including 3C286, 3C48, 3C147, and several planets, moons, and asteroids.

Example: `[3.06,0.0,0.0,0.0]`

- **spix** -- Spectral index for fluxdensity  $S = \text{fluxdensity} * (\text{freq}/\text{reffreq})^{\text{spix}}$  Only used if fluxdensity is being used. If fluxdensity is positive, and spix is nonzero, then reffreq must be set too. It is applied in the same way to all polarizations, and does not account for Faraday rotation or depolarization.

- **scalebychan** -- This determines whether the fluxdensity set in the model is calculated on a per channel basis. If False then only one fluxdensity value is calculated per spw.

- **standard** -- Flux density standard, used if fluxdensity[0] less than 0.0. The options are: 'Baars','Perley 90','Perley-Taylor 95', 'Perley-Taylor 99', 'Perley-Butler 2010' and 'Butler-JPL-Horizons 2010'. default: 'Butler-JPL-Horizons 2012' for solar system object 'Perley-Butler 2010' otherwise

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Set the model flux densities for all the amplitude calibrators:

```
>>> hif_setjy()
```

## 2.17 pipeline.hif.cli.hif\_setmodels

**hif\_setmodels**(vis=None, reference=None, refintent=None, transfer=None, transintent=None, reffile=None, normfluxes=None, scalebychan=None, parallel=None) → ResultsList[Results]

Set calibrator source models.

Set model fluxes values for calibrator reference and transfer sources using lookup values. By default the reference sources are the flux calibrators and the transfer sources are the bandpass, phase, and check source calibrators. Reference sources which are also in the transfer source list are removed from the transfer source list.

Built-in lookup tables are used to compute models for solar system object calibrators. Point source models are used for other calibrators with flux densities provided in the reference file. Normalized fluxes are computed for transfer sources if the **normfluxes** parameter is set to True.

The default reference file is 'flux.csv' in the current working directory. This file is usually created in the importdata stage. The file is in 'csv' format and contains the following comma delimited columns.

vis,fieldid,spwid,I,Q,U,V,pix,comment

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.  
Example: `['M32A.ms', 'M32B.ms']`
- **reference** -- A string containing a comma delimited list of field names defining the reference calibrators. Defaults to field names with intent 'AMPLITUDE'.  
Example: `'M82,3C273'`
- **refintent** -- A string containing a comma delimited list of intents used to select the reference calibrators. Defaults to 'AMPLITUDE'.  
Example: `'BANDPASS'`
- **transfer** -- A string containing a comma delimited list of field names defining the transfer calibrators. Defaults to field names with intent ".  
Example: `'J1328+041,J1206+30'`
- **transintent** -- A string containing a comma delimited list of intents defining the transfer calibrators. Defaults to `'BANDPASS,PHASE,CHECK'`. '' stands for no transfer sources.  
Example: `'PHASE'`

- **reffile** -- The reference file containing a lookup table of point source models This file currently defaults to 'flux.csv' in the working directory. This file must conform to the standard pipeline 'flux.csv' format  
Example: `'myfluxes.csv'`
- **normfluxes** -- Normalize the transfer source flux densities.
- **scalebychan** -- Scale the flux density on a per channel basis or else on a per spw basis
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`  
Default: `None` (equivalent to `False`)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Set model fluxes for the flux, bandpass, phase, and check sources.

```
>>> hif_setmodels()
```

## 2.18 pipeline.hif.cli.hif\_transformimagedata

```
hif_transformimagedata(vis=None, outputvis=None, field=None, intent=None, spw=None, datacolumn=None, chanbin=None, timebin=None, replace=None, clear_pointing=None, modify_weights=None, wtmode=None) → ResultsList[Results]
```

Extract fields for the desired VLASS image to a new MS and reset weights if desired.

**Parameters**

- **vis** -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSs, or tar files of MSs, If ASDM files are specified, they will be converted to MS format.  
Example: `vis=['X227.ms', 'asdms.tar.gz']`
- **outputvis** -- The output MeasurementSet.
- **field** -- Set of data selection field names or ids, `' '` for all.
- **intent** -- Set of data selection intents, `' '` for all.
- **spw** -- Set of data selection spectral window ids `' '` for all.
- **datacolumn** -- Select spectral windows to split. The standard CASA options are supported  
Example: `'data', 'model'`
- **chanbin** -- Bin width for channel averaging.
- **timebin** -- Bin width for time averaging.
- **replace** -- If a split was performed delete the parent MS and remove it from the context. example: True or False
- **clear\_pointing** -- Clear the pointing table.
- **modify\_weights** -- Re-initialize the weights.
- **wtmode** -- optional weight initialization mode when `modify_weights=True`

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Basic transformimagedata task

```
>>> hif_transformimagedata()
```

## 2.19 pipeline.hif.cli.hif\_uvcontsub

**hif\_uvcontsub**(vis=None, field=None, intent=None, spw=None, fitorder=None, parallel=None) → ResultsList[Results]

Fit and subtract continuum from the data.

hif\_uvcontsub fits the continuum for the frequency ranges given in the cont.dat file, subtracts that fit from the uv data and generates a new set of MSes containing the continuum subtracted (i.e. line) data. The fit is attempted for all science targets and spws. If a fit is impossible, the corresponding data selection is not written to the output line MS.

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the <hifa,hifv>\_importdata task. ": use all MeasurementSets in the context  
Examples: 'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']
- **field** -- The list of field names or field ids for which UV continuum fits are computed. Defaults to all fields.  
Examples: '3C279', '3C279,M82'
- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched.  
": Defaults to all data with TARGET intent.
- **spw** -- The list of spectral windows and channels for which uv continuum fits are computed. ", Defaults to all science spectral windows.  
Example: '11,13,15,17'
- **fitorder** -- Polynomial order for the continuum fits per source and spw. Defaults to {} which means fit order 1 for all sources and spws. If an explicit dictionary is given then all unspecified selections still default to 1.  
Example: {'3C279': {'15': 1, '17': 2}, 'M82': {'13': 2}}
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: 'automatic', 'true', 'false', True, False  
Default: None (equivalent to False)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Fit and subtract continuum for all science targets and spws

```
>>> hif_uvcontsub()
```

2. Fit and subtract continuum only for a subset of fields

```
>>> hif_uvcontsub(field='3C279,M82')
```

3. Fit and subtract continuum only for a subset of spws

```
>>> hif_uvcontsub(spw='11,13')
```

4. Override automatic fit order choice

```
>>> hif_uvcontsub(fitorder={'3C279': {'15': 1, '17': 2}, 'M82': {'13': 2}})
```

## PIPELINE.HIFA.CLI

Interferometry ALMA Tasks

### Functions

<code>hifa_antpos</code>	Derive antenna position calibration tables for a list of MeasurementSets.
<code>hifa_bandpass</code>	Compute bandpass calibration solutions.
<code>hifa_bandpassflag</code>	Bandpass calibration flagging.
<code>hifa_bpsolint</code>	Compute optimal bandpass calibration solution intervals.
<code>hifa_diffgaincal</code>	Derive SpW phase offsets from differential gain calibrator.
<code>hifa_exportdata</code>	Prepare interferometry data for export.
<code>hifa_flagdata</code>	Do metadata based flagging of a list of MeasurementSets.
<code>hifa_flagtargets</code>	Do science target flagging.
<code>hifa_fluxcalflag</code>	Locate and flag line regions in solar system flux calibrators.
<code>hifa_gaincalsnr</code>	Compute gaincal signal-to-noise ratios per spw.
<code>hifa_gfluxscale</code>	Derive flux density scales from standard calibrators.
<code>hifa_gfluxscaleflag</code>	Flag the flux, diffgain, phase calibrators and check source.
<code>hifa_imageprecheck</code>	Calculates the best Briggs robust parameter to achieve sensitivity and angular resolution goals.
<code>hifa_importdata</code>	Imports data into the interferometry pipeline.
<code>hifa_lock_refant</code>	Lock reference antenna list.
<code>hifa_polcal</code>	Derive instrumental polarization calibration for ALMA.
<code>hifa_polcalflag</code>	Flag polarization calibrators.
<code>hifa_renorm</code>	ALMA renormalization.
<code>hifa_restoredata</code>	Restore flagged and calibration interferometry data from a pipeline run.
<code>hifa_session_refant</code>	Select best reference antenna for session(s).
<code>hifa_spwphaseup</code>	Compute phase calibration spw map and per spw phase offsets.
<code>hifa_targetflag</code>	Flag target source outliers.
<code>hifa_timegaincal</code>	Determine temporal gains from calibrator observations.
<code>hifa_tsysflag</code>	Flag deviant system temperatures for ALMA interferometry measurements.
<code>hifa_tsysflagcontamination</code>	Flag line contamination in ALMA interferometric Tsys caltables.
<code>hifa_unlock_refant</code>	Unlock reference antenna list.
<code>hifa_wvrgcal</code>	Generate a gain table based on Water Vapor Radiometer (WVR) data.
<code>hifa_wvrgcalflag</code>	Flag bad WVR calibration in gain table and interpolate over antennas with bad radiometers.

## 3.1 pipeline.hifa.cli.hifa\_antpos

**hifa\_antpos**(vis: list[str] | None = None, caltable: list[str] | None = None, hm\_antpos: Literal['online', 'manual', 'file'] | None = None, antenna: str | None = None, offsets: list[float] | None = None, antposfile: str | None = None, threshold: float | None = None, snr: float | None = None, search: Literal['both\_latest', 'both\_closest'] | None = None) → ResultsList[Results]

Derive antenna position calibration tables for a list of MeasurementSets.

The *hifa\_antpos* task corrects antenna positions recorded in the ASDMs using updated calibration information obtained after the observation. Corrections can be input by hand, read from a file on disk, or by querying an ALMA database service.

The *antposfile* parameter serves a dual purpose, depending on which mode is set.

For *hm\_antpos='file'*, *antposfile* defines the antenna positions file in 'csv' format containing 6 comma-delimited columns as shown below. This file should not include blank lines, including after the end of the last entry. This parameter is required for *hm\_antpos='file'*.

Example of contents for a .csv file:

```
ms, antenna, xoffset, yoffset, zoffset, comment
uid__A002_X30a93d_X43e.ms, DV11, 0.000, 0.010, 0.000, 'No comment'
uid__A002_X30a93d_X43e.dup.ms, DV11, 0.000, -0.010, 0.000, 'No comment'
```

The offset values in this file are in meters.

For *hm\_antpos='online'*, *antposfile* defines the base outfile name used by the CASA tasks *getantposalma* and *gencal* with the MS basename prepended to it. The file must be in JSON format. If no value is set, it will default to *antennapos.json*.

The corrections are used to generate a calibration table which is recorded in the pipeline context and applied to the raw visibility data, on the fly to generate other calibration tables, or permanently to generate calibrated visibilities for imaging.

### Parameters

- **vis** -- List of input MeasurementSets. Defaults to those specified in the pipeline context.

Example: ['ngc5921.ms']

- **caltable** -- List of output calibration table names. Defaults to the standard pipeline naming convention.

Example: ['ngc5921.gcal']

- **hm\_antpos** --

– 'online' : Query ALMA database through CASA task *getantposalma* or reuse

pre-existing queried/downloaded JSON files. Files follow the naming pattern *{eb\_name}.{antposfile}*.

For multi-MS pipeline runs, the MS basename is appended to the filename (e.g., *uid\_\_A002\_X123\_X4567.antennapos.json*). – 'manual' : Use user-provided corrections. – 'file' : Load corrections from a single old-style CSV antenna position file.

Example: 'manual'

- **antenna** -- A comma-separated string of antennas whose positions are to be corrected (if *hm\_antpos* is 'manual' or 'online').

Example: 'DV05,DV07'

- **offsets** -- A flat list of floating-point offsets (X, Y, Z) for all specified antennas. The length of the list must be three times the number of antennas.

Example (for two antennas): [0.01, 0.02, 0.03, 0.03, 0.02, 0.01]

- **antposfile** --

**Path to a csv file containing antenna position offsets for *hm\_antpos='file'* (required) or the name**

of the outfile created by *getantposalma* for *hm\_antpos='online'*. In order to work with multi-MS pipeline runs, the MS basename will be appended to the file name when using *hm\_antpos='online'* (i.e. '*uid\_\_A002\_X123\_X4567.antennapos.json*').

Default: 'antennapos.json'

- **threshold** -- Threshold value (in wavelengths) above which antenna position offsets are highlighted in the weblog. Defaults to 1.0.

Example: 1.0

- **snr** -- A float value describing the signal-to-noise threshold used by the *getantposalma* task. Antennas with snr below the threshold will not be retrieved. Only used with *hm\_antpos='online'*. Defaults to 'default'.

Example: 5.0

- **search** -- Search algorithm used by the *getantposalma* task. Supports 'both\_latest', 'both\_closest', and 'auto'. Only used with *hm\_antpos='online'*. Defaults to 'auto'.

Example: 'both\_closest'

### Returns

The results object for the pipeline task is returned.

### Examples

1. Correct the position of antenna 'DV05' for all the visibility files in a single pipeline run:

```
>>> hifa_antpos(antenna='DV05', offsets=[0.01, 0.02, 0.03])
```

2. Correct the position of antennas for all the visibility files in a single pipeline run using antenna positions files on disk. These files are assumed to conform to a default naming scheme if **antposfile** is unspecified by the user:

```
>>> hifa_antpos(hm_antpos='file', antposfile='myantposfile.csv')
```

3. Correct the position of antennas for all the visibility files in a single pipeline run using antenna positions retrieved from DB, limiting the selection to antennas with S/N of 5.0 or more and using the 'both\_closest' search algorithm. A JSON file is returned and fed into the gencal task to apply corrections.

```
>>> hifa_antpos(hm_antpos='online', snr=5.0, search='both_closest')
```

## 3.2 pipeline.hifa.cli.hifa\_bandpass

**hifa\_bandpass**(vis=None, caltable=None, field=None, intent=None, spw=None, antenna=None, hm\_phaseup=None, phaseupbw=None, phaseupmaxsolint=None, phaseupsolint=None, phaseupsnr=None, phaseupsols=None, hm\_phaseup\_combine=None, hm\_bandpass=None, solint=None, maxchannels=None, evenbpints=None, bpsnr=None, minbpsnr=None, bpsols=None, combine=None, refant=None, solnorm=None, minblperant=None, minsnr=None, unregister\_existing=None, hm\_auto\_fillgaps=None, parallel=None) → ResultsList[Results]

Compute bandpass calibration solutions.

The hifa\_bandpass task computes a bandpass solution for every specified science spectral window. By default, a 'phaseup' pre-calibration is performed and applied on the fly to the data, before the bandpass is computed.

The hif\_refant task may be used to pre-compute a prioritized list of reference antennas.

### Parameters

- **vis** -- List of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['ngc5921.ms']`

- **caltable** -- List of names for the output calibration tables. Defaults to the standard pipeline naming convention.

Example: `caltable=['ngc5921.gcal']`

- **field** -- The list of field names or field ids for which bandpasses are computed. Set to `field=""` by default, which means the task will select all fields.

Example: `field='3C279', field='3C279,M82'`

- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched. Set to `intent=""` by default, which means the task will select all data with the BANDPASS intent.

Example: `intent='*PHASE*'`

- **spw** -- The list of spectral windows and channels for which bandpasses are computed. Set to `spw=""` by default, which means the task will select all science spectral windows.

Example: `spw='11,13,15,17'`

- **antenna** -- Set of data selection antenna IDs

- **hm\_phaseup** -- The pre-bandpass solution phaseup gain heuristics. The options are:

- `'snr'`: compute solution required to achieve the specified SNR
- `'manual'`: use manual solution parameters
- `''`: skip phaseup

Example: `hm_phaseup='manual'`

- **phaseupbw** -- Bandwidth to be used for phaseup. Used when `hm_phaseup= 'manual'`.

Example:

- `phaseupbw=''` to use entire bandpass
- `phaseupbw='500MHz'` to use central 500MHz

- **phaseupmaxsolint** -- Maximum phase correction solution interval (in seconds) allowed in very low-SNR cases. Used only when `hm_phaseup='snr'`.

Example: `phaseupmaxsolint=60.0`

- **phaseupsolint** -- The phase correction solution interval in CASA syntax. Used when `hm_phaseup='manual'` or as a default if the `hm_phaseup='snr'` heuristic computation fails.

Example: `phaseupsolint='300s'`

- **phaseupsnr** -- The required SNR for the phaseup solution. Used to calculate the phaseup time solint, and only if `hm_phaseup='snr'`.

Example: `phaseupsnr=10.0`

- **phaseupnsols** -- The minimum number of phaseup gain solutions. Used only if `hm_phaseup='snr'`.

Example: `phaseupnsols=4`

- **hm\_phaseup\_combine** -- The spw combination heuristic for the phase-up solution. Accepts one of following 3 options:

- `'snr'`, default: heuristics will use `combine='spw'` in `phase-up gaincal`, when SpWs have  $\text{SNR} < \text{phaseupsnr}$  (default = 20).

- '**always**': heuristic will force combine='spw' in the phase-up gaincal.
- '**never**': heuristic will not use spw combination; this was the default logic for Pipeline release 2024 and prior.

Example: `hm_phaseup_combine='always'`

- **hm\_bandpass** -- The bandpass solution heuristics. The options are: '**snr**': compute the solution required to achieve the specified SNR '**smoothed**': simple 'smoothing' i.e. spectral solint>1chan '**fixed**': use the user defined parameters for all spws

Example: `hm_bandpass='snr'`

- **solint** -- Time and channel solution intervals in CASA syntax. Default is solint='inf', which is used when **hm\_bandpass='fixed'**. If **hm\_bandpass='snr'**, then the task will attempt to compute and use an optimal SNR-based solint (and warn if this solint is not good enough). If **hm\_bandpass='smoothed'**, the task will override the spectral solint with bandwidth/maxchannels.

Example: `solint='int'`

- **maxchannels** -- The bandpass solution 'smoothing' factor in channels, i.e. spectral solint will be set to bandwidth / maxchannels Set to 0 for no smoothing. Used if **hm\_bandpass='smoothed'**.

Example: `maxchannels=240`

- **evenbpints** -- Force the per spw frequency solint to be evenly divisible into the spw bandpass if **hm\_bandpass='snr'**.

Example: `evenbpints=False`

- **bpsnr** -- The required SNR for the bandpass solution. Used only if **hm\_bandpass='snr'**.

Example: `bpsnr=30.0`

- **minbpsnr** -- The minimum required SNR for the bandpass solution when strong atmospheric lines exist in Tsys spectra. Used only if **hm\_bandpass='snr'**.

Example: `minbpsnr=10.0`

- **bponsols** -- The minimum number of bandpass solutions. Used only if **hm\_bandpass='snr'**.

Example: `bponsols=8`

- **combine** -- Data axes to combine for solving. Axes are '' , 'scan' , 'spw' , 'field' or any comma-separated combination.

Example: `combine='scan,field'`

- **refant** -- List of reference antenna names. Defaults to the value(s) stored in the pipeline context. If undefined in the pipeline context defaults to the CASA reference antenna naming scheme.

Example: `refant='DV06,DV07'`

- **solnorm** -- Normalise the bandpass solution; defaults to **True**.

Example: `solnorm=False`

- **minblperant** -- Minimum number of baselines required per antenna for each solve. Antennas with fewer baselines are excluded from solutions.

Example: `minblperant=4`

- **minsnr** -- Solutions below this SNR are rejected in the phaseup and bandpass solves.

Example: `minsnr=3.0`

- **unregister\_existing** -- Unregister all bandpass calibrations from the pipeline context before registering the new bandpass calibrations from this task. Defaults to False.

Example: `unregister_existing=True`

- **hm\_auto\_fillgaps** -- If True, then the `hm_bandpass` = 'snr' or 'smoothed' modes, that solve bandpass per SpW, are performed with CASA bandpass task parameter 'fillgaps' set to a quarter of the respective SpW bandwidth (in channels). If False, then these bandpass solves will use fillgaps=0. The `hm_bandpass='fixed'` mode is unaffected by `hm_auto_fillgaps` and always uses fillgaps=0.

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`

Default: `None` (equivalent to `False`)

### Returns

The results object for the pipeline task is returned.

### Examples

1. Compute a channel bandpass for all visibility files in the pipeline context using the CASA reference antenna determination scheme:

```
>>> hifa_bandpass()
```

2. Same as the above but precompute a prioritized reference antenna list:

```
>>> hif_refant()
>>> hifa_bandpass()
```

## 3.3 pipeline.hifa.cli.hifa\_bandpassflag

```
hifa_bandpassflag(vis=None, ctable=None, intent=None, field=None, spw=None, antenna=None, hm_phaseup=None, phaseupbw=None, phaseupmaxsolint=None, phaseupsolint=None, phaseupsnr=None, phaseupsols=None, hm_phaseup_combine=None, hm_bandpass=None, solint=None, maxchannels=None, evenbpints=None, bpsnr=None, minbpsnr=None, bpnols=None, combine=None, refant=None, minblperant=None, minsnr=None, solnorm=None, antnegsig=None, antpossig=None, tmantint=None, tmint=None, tmbl=None, antblnegsig=None, antblpossig=None, relaxed_factor=None, niter=None, hm_auto_fillgaps=None, parallel=None) → ResultsList[Results]
```

Bandpass calibration flagging.

This task performs a preliminary phased-up bandpass solution and temporarily applies it, then computes the flagging heuristics by calling `hif_correctedampflag` which looks for outlier visibility points by statistically examining the scalar difference of the corrected amplitudes minus model amplitudes, and then flags those outliers. The philosophy is that only outlier data points that have remained outliers after calibration will be flagged. Note that the phase of the data is not assessed.

Plots are generated at two points in this workflow: after bandpass calibration but before flagging heuristics are run, and after flagging heuristics have been run and applied. If no points were flagged, the 'after' plots are not generated or displayed.

### Parameters

- **vis** -- List of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['ngc5921.ms']`

- **ctable** -- List of names for the output calibration tables. Defaults to the standard pipeline naming convention.

Example: `caltable=['ngc5921.gcal']`

- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched. Set to `intent=''` by default, which means the task will select all data with the BANDPASS intent.

Example: `intent='*PHASE*'`

- **field** -- The list of field names or field ids for which bandpasses are computed. Set to `field=""` by default, which means the task will select all fields.

Example: `field='3C279', field='3C279,M82'`

- **spw** -- The list of spectral windows and channels for which bandpasses are computed. Set to `spw=""` by default, which means the task will select all science spectral windows.

Example: `spw='11,13,15,17'`

- **antenna** -- Set of data selection antenna IDs

- **hm\_phaseup** -- The pre-bandpass solution phaseup gain heuristics. The options are:

- `'snr'`: compute solution required to achieve the specified SNR
- `'manual'`: use manual solution parameters
- `''`: skip phaseup

Example: `hm_phaseup='manual'`

- **phaseupbw** -- Bandwidth to be used for phaseup. Used when `hm_phaseup='manual'`.

Example:

- `phaseupbw=''` to use entire bandpass
- `phaseupbw='500MHz'` to use central 500MHz

- **phaseupmaxsolint** -- Maximum phase correction solution interval (in seconds) allowed in very low-SNR cases. Used only when `hm_phaseup='snr'`.

Example: `phaseupmaxsolint=60.0`

- **phaseupsolint** -- The phase correction solution interval in CASA syntax. Used when `hm_phaseup='manual'` or as a default if the `hm_phaseup='snr'` heuristic computation fails.

Example: `phaseupsolint='300s'`

- **phaseupsnr** -- The required SNR for the phaseup solution. Used to calculate the phaseup time solint, and only if `hm_phaseup='snr'`.

Example: `phaseupsnr=10.0`

- **phaseupnsols** -- The minimum number of phaseup gain solutions. Used only if `hm_phaseup='snr'`.

Example: `phaseupnsols=4`

- **hm\_phaseup\_combine** -- The spw combination heuristic for the phase-up solution. Accepts one of following 3 options:

- `'snr'`, default: heuristics will use `combine='spw'` in phase-up gaincal when SpWs have  $\text{SNR} < \text{phaseupsnr}$  (default = 20).
- `'always'`: heuristic will force `combine='spw'` in the phase-up gaincal.
- `'never'`: heuristic will not use spw combination; this was the default logic for Pipeline release 2024 and prior.

Example: `hm_phaseup_combine='always'`

- **hm\_bandpass** -- The bandpass solution heuristics. The options are: '**snr**': compute the solution required to achieve the specified SNR '**smoothed**': simple 'smoothing' i.e. spectral solint>1chan '**fixed**': use the user defined parameters for all spws
- **solint** -- Time and channel solution intervals in CASA syntax. Default is solint='inf', which is used when **hm\_bandpass='fixed'**. If **hm\_bandpass='snr'**, then the task will attempt to compute and use an optimal SNR-based solint (and warn if this solint is not good enough). If **hm\_bandpass='smoothed'**, the task will override the spectral solint with bandwidth/maxchannels.
- **maxchannels** -- The bandpass solution 'smoothing' factor in channels, i.e. spectral solint will be set to bandwidth/maxchannels Set to 0 for no smoothing. Used if **hm\_bandpass= 'smoothed'**.
 

Example: `maxchannels=240`
- **evenbpints** -- Force the per spw frequency solint to be evenly divisible into the spw bandpass if **hm\_bandpass= 'snr'**.
 

Example: `evenbpints=False`
- **bpsnr** -- The required SNR for the bandpass solution. Used only if **hm\_bandpass='snr'**.
 

Example: `bpsnr=30.0`
- **minbpsnr** -- The minimum required SNR for the bandpass solution when strong atmospheric lines exist in Tsys spectra. Used only if **hm\_bandpass='snr'**.
 

Example: `minbpsnr=10.0`
- **bpnsols** -- The minimum number of bandpass solutions. Used only if **hm\_bandpass='snr'**.
 

Example: `bpnsols=8`
- **combine** -- Data axes to combine for solving. Axes are '' , 'scan' , 'spw' , 'field' or any comma-separated combination.
 

Example: `combine='scan,field'`
- **refant** -- List of reference antenna names. Defaults to the value(s) stored in the pipeline context. If undefined in the pipeline context defaults to the CASA reference antenna naming scheme.
 

Example: `refant='DV06,DV07'`
- **minblperant** -- Minimum number of baselines required per antenna for each solve. Antennas with fewer baselines are excluded from solutions.
 

Example: `minblperant=4`
- **minsnr** -- Solutions below this SNR are rejected.
 

Example: `minsnr=3.0`
- **solnorm** -- Normalise the bandpass solution; defaults to **True**.
- **antnegsig** -- Lower sigma threshold for identifying outliers as a result of bad antennas within individual timestamps.
 

Example: `antnegsig=4.0`
- **antpossig** -- Upper sigma threshold for identifying outliers as a result of bad antennas within individual timestamps.
 

Example: `antpossig=4.6`
- **tmantint** -- Threshold for maximum fraction of timestamps that are allowed to contain outliers.
 

Example: `tmantint=0.063`

- **tmint** -- Initial threshold for maximum fraction of 'outlier timestamps' over 'total timestamps' that a baseline may be a part of.  
Example: `tmint=0.085`
- **tmb1** -- Initial threshold for maximum fraction of 'bad baselines' over 'all baselines' that an antenna may be a part of.  
Example: `tmb1=0.175`
- **antblnegsig** -- Lower sigma threshold for identifying outliers as a result of 'bad baselines' and/or 'bad antennas' within baselines (across all timestamps).  
Example: `antblnegsig=3.4`
- **antblpossig** -- Upper sigma threshold for identifying outliers as a result of 'bad baselines' and/or 'bad antennas' within baselines (across all timestamps).  
Example: `antblpossig=3.2`
- **relaxed\_factor** -- Relaxed value to set the threshold scaling factor to under certain conditions (see documentation of the underlying correctedampflag task).  
Example: `relaxed_factor=2.0`
- **niter** -- Maximum number of times to iterate on evaluation of flagging heuristics. If an iteration results in no new flags, then subsequent iterations are skipped.  
Example: `niter=2`
- **hm\_auto\_fillgaps** -- If True, then the `hm_bandpass='snr'` or 'smoothed' modes, that solve bandpass per SpW, are performed with CASA bandpass task parameter 'fillgaps' set to a quarter of the respective SpW bandwidth (in channels). If False, then these bandpass solves will use fillgaps=0. The `hm_bandpass='fixed'` mode is unaffected by `hm_auto_fillgaps` and always uses fillgaps=0.
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: `'automatic', 'true', 'false', True, False`  
Default: `None` (equivalent to `False`)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. run with recommended settings to create bandpass solution with flagging using recommended thresholds:

```
>>> hifa_bandpassflag()
```

## 3.4 pipeline.hifa.cli.hifa\_bpsolint

```
hifa_bpsolint(vis=None, field=None, intent=None, spw=None, phaseupsnr=None, minphaseupints=None, evenpints=None,
               bpsnr=None, minbpsnr=None, minbpnchan=None, hm_nantennas=None, maxfracflagged=None) →
ResultsList[Results]
```

Compute optimal bandpass calibration solution intervals.

The task estimates the optimal time and frequency solution intervals needed to achieve the required signal-to-noise ratio. This estimation is based on nominal ALMA array characteristics and the observation's metadata.

The phaseup gain time and bandpass frequency intervals are determined as follows:

- For each data set the list of source(s) to use for bandpass solution signal-to-noise estimation is compiled based on the values of the `field`, `intent`, and `spw` parameters.
- Source fluxes are determined for each spw and source combination.
- Fluxes in Jy are derived from the pipeline context.
- Pipeline context fluxes are derived from the online flux calibrator catalog, the ASDM, or the user via the `flux.csv` file.
- If no fluxes are available the task terminates.
- Atmospheric calibration and observations scans are determined for each spw and source combination.
- If `intent` is set to 'PHASE' and there are no atmospheric scans associated with the 'PHASE' calibrator, 'TARGET' atmospheric scans will be used instead.
- If atmospheric scans cannot be associated with any of the spw and source combinations the task terminates.
- Science spws are mapped to atmospheric spws for each science spw and source combination.
- If mappings cannot be determined for any of the spws the task terminates.
- The median Ts<sub>ys</sub> value for each atmospheric spw and source combination is determined from the SYSCAL table. Medians are computed first by channel, then by antenna, in order to reduce sensitivity to deviant values.
- The science spw parameters, exposure time(s), and integration time(s) are determined.
- The phase-up time interval, in time units and number of integrations required to meet the `phaseupsnr` are computed, along with the sensitivity in mJy and the signal-to-noise per integration. Nominal Ts<sub>ys</sub> and sensitivity values per receiver band provided by the ALMA project are used for this estimate.
- Warnings are issued if estimated phase-up gain time solution would contain fewer than `minphaseupints` solutions.
- The frequency interval, in MHz and number of channels required to meet the `bpsnr` are computed, along with the per channel sensitivity in mJy and the per channel signal-to-noise. Nominal Ts<sub>ys</sub> and sensitivity values per receiver band provided by the ALMA project are used for this estimate.
- Warnings are issued if estimated bandpass solution would contain fewer than `minbpnchan` solutions.
- If strong atmospheric features are detected in the Ts<sub>ys</sub> spectrum for a given spw, the frequency interval of bandpass solution is recalculated to meet the lower threshold `minbpsnr`, i.e., a lower snr is tolerated in order to preserve enough frequency intervals to capture the atmospheric line.

## Parameters

- `vis` -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.  
Example: `vis=['M82A.ms', 'M82B.ms']`
- `field` -- The list of field names of sources to be used for signal-to-noise estimation. Defaults to all fields with the standard intent.  
Example: `field='3C279'`
- `intent` -- A string containing a comma delimited list of intents against which the selected fields are matched. Defaults to 'BANDPASS'.  
Example: `intent='PHASE'`
- `spw` -- The list of spectral windows and channels for which gain solutions are computed. Defaults to all the science spectral windows for which there are both 'intent' and TARGET intents.  
Example: `spw='13,15'`

- **phaseupsnr** -- The required phase-up gain time interval solution signal-to-noise.  
Example: `phaseupsnr=10.0`
- **minphaseupints** -- The minimum number of time intervals in the phase-up gain solution.  
Example: `minphaseupints=4`
- **evenbpints** -- Use a bandpass frequency solint that is an integer divisor of the spw bandwidth, to prevent the occurrence of one narrower fractional frequency interval.
- **bpsnr** -- The required bandpass frequency interval solution signal-to-noise.  
Example: `bpsnr=30.0`
- **minbpsnr** -- The minimum required bandpass frequency interval solution signal-to-noise when strong atmospheric lines exist in Tsys spectra.  
Example: `minbpsnr=10.0`
- **minbpnchan** -- The minimum number of frequency intervals in the bandpass solution.  
Example: `minbpnchan=16`
- **hm\_nantennas** -- The heuristics for determines the number of antennas to use in the signal-to-noise estimate. The options are '`all`' and '`unflagged`'. The '`unflagged`' option is not currently supported.  
Example: `hm_nantennas='unflagged'`
- **maxfracflagged** -- The maximum fraction of an antenna that can be flagged before it is excluded from the signal-to-noise estimate.  
Example: `maxfracflagged=0.80`

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Estimate the phaseup gain time interval and the bandpass frequency interval required to match the desired signal-to-noise for bandpass solutions:

```
>>> hifa_bpsolint()
```

## 3.5 pipeline.hifa.cli.hifa\_diffgaincal

```
hifa_diffgaincal(vis=None, flagging_frac_limit=None, hm_spwmapmode=None, missing_scans_frac_limit=None) →
    ResultsList[DiffGaincalResults]
```

Derive SpW phase offsets from differential gain calibrator.

This task creates the spectral window phase offset table used to allow calibrating the "on-source" spectral setup with phase gains from a "reference" spectral setup. Currently this setup with two different SpectralSpecs is used by the band-to-band mode, for a high and low frequency band.

A bright point source quasar, called the Differential Gain Calibrator (DIFFGAIN) source, is used for this purpose. This DIFFGAIN source is typically observed in groups of interleaved "reference" and "on-source" scans. These blocks typically occur once at the start and once at the end of the observation. In very long observations, there may be a group of scans occurring during the middle.

The procedure for the phase calibration process is as follow:

- **reference** calibration uses the "reference" spectral setup, the "low" frequency for band-to-band observations. The solint is fixed as 'inf' (i.e. solutions are made per scan). Spectral window combination is governed by the hm\_spwmapmode options described below. In standard pipeline operation **auto** is used - basing spw combination upon SNR and flagging level. These reference phase solutions are later applied on-the-fly while solving the (band-to-band) **phase offset**. The premise being that the **reference** solutions correct for atmospheric phase variability.
- **phase offset** (i.e. the band-to-band correction) calibration uses the "on-source" spectral setup, the "high" frequency for band-to-band observations. The above **reference** phase corrections are applied on-the-fly using a linearPD interpolation - this corrects (scales) the phases according to the different ratio of the frequency bands. The solint = 'inf' and each group of scan blocks (typically at the start and end of the observation) are combined respectively. The phase offset solution generally comprises of 2 time solutions per spw, per polarization. Spectral window combination is governed by the hm\_spwmapmode options described below. **auto** is used - basing spw combination upon SNR and flag data. These solutions are stored in the pipeline context for later application to the TARGET and CHECK intent(s).
- **residual offset** solutions are produced by applying both **reference** and band-to-band **phase offset** solutions to the 'on-source' DIFFGAIN intent on-the-fly, and subsequently solving phases per spw, per scan using solint='inf'. Spectral window combination is governed by the hm\_spwmapmode options described below. **auto** is used - basing spw combination upon SNR and flagging level.

## Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `['M32A.ms', 'M32B.ms']`

- **flagging\_frac\_limit** -- if the fraction of flagged data in the temporary phase gaintable exceeds this limit then SpW combination is triggered.

- **hm\_spwmapmode** -- The spectral window mapping heuristic mode. The options are:

- **'all'**: SpW combination is forced for the diffgain

low-frequency reference intent solutions, the diffgain high-frequency source intent solutions (actual band-to-band offsets), and for the diagnostic residual phase offsets on the diffgain high-frequency source intent.

- **'auto'**: Assess need for SpW combination based on SpwMapping

from hifa\_spwphaseup, and where necessary check the gaintable for missing SpWs / too many flagged data / too few scan solutions.

- **'both'**: SpW combination is forced for the diffgain

low-frequency reference intent solutions and for the diagnostic residual phase offsets on the diffgain high-frequency source intent.

- **'offset'**: SpW combination is forced for the diffgain

high-frequency source intent solutions (actual band-to-band offsets).

- **'reference'**: SpW combination is forced for the diffgain

low-frequency reference intent solutions.

- **'residual'**: SpW combination is forced for the diagnostic

residual phase offsets on the diffgain high-frequency source intent.

Example: `hm_spwmapmode='auto'`

- **missing\_scans\_frac\_limit** -- if the fraction of missing scans in the temporary phase gaintable exceeds this limit then SpW combination is triggered.

## Returns

The results object for the pipeline task is returned.

## Examples

- Derive SpW phase offsets from differential gain calibrator.

```
>>> hifa_diffgaincal()
```

## 3.6 pipeline.hifa.cli.hifa\_exportdata

**hifa\_exportdata**(vis: list[str] = None, session: list[str] = None, imaging\_products\_only: bool = None, exportmses: bool = None, tarms: bool = None, pprfile: list[str] = None, calintents: str = None, calimages: list[str] = None, targetimages: list[str] = None, products\_dir: str = None) → Results

Prepare interferometry data for export.

The hifa\_exportdata task for ALMA CASA pipeline exports the data defined in the pipeline context and exports it to the data products directory, converting and or packing it as necessary.

The current version of the task exports the following products

- an XML file containing the pipeline processing request
- a tar file per ASDM / MS containing the final flags version
- a text file per ASDM / MS containing the final calibration apply list
- a FITS image for each selected calibrator source image
- a FITS image for each selected science target source image
- a tar file per session containing the caltables for that session
- a tar file containing the file web log
- a text file containing the final list of CASA commands
- an XML "manifest" file listing the products
- an XML "aquareport" file listing the QA scores and sub-scores, image sensitivities, and other numerical information

### Parameters

- vis** -- List of visibility data files for which flagging and calibration information will be exported. Defaults to the list maintained in the pipeline context. Example: `vis=['X227.ms', 'X228.ms']`
- session** -- List of sessions one per visibility file. Currently defaults to a single virtual session containing all the visibility files in vis. In the future, this will default to the set of observing sessions defined in the context. Example: `session=['session1', 'session2']`
- imaging\_products\_only** -- Export science target imaging products only
- exportmses** -- Export the final MeasurementSets instead of the final flags, calibration tables, and calibration instructions.
- tarms** -- Tar final MeasurementSets
- pprfile** -- Name of the pipeline processing request to be exported. Defaults to a file matching the template 'PPR\_\*.xml'. Example: `pprfile=['PPR_GRB021004.xml']`
- calintents** -- List of calibrator image types to be exported. Defaults to all standard calibrator intents, 'BANDPASS', 'PHASE', 'FLUX'. Example: `'PHASE'`
- calimages** -- List of calibrator images to be exported. Defaults to all calibrator images recorded in the pipeline context. Example: `calimages=['3C454.3.bandpass', '3C279.phase']`

- **targetimages** -- List of science target images to be exported. Defaults to all science target images recorded in the pipeline context. Example: `targetimages=['NGC3256.band3', 'NGC3256.band6']`
- **products\_dir** -- Name of the data products subdirectory. Defaults to `'.'`. Example: `products_dir='.. / products'`

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Export the pipeline results for a single session to the data products directory:

```
>>> !mkdir .. / products
>>> hif_exportdata(products_dir='.. / products')
```

2. Export the pipeline results to the data products directory specify that only the gain calibrator images be saved:

```
>>> !mkdir .. / products
>>> hif_exportdata(products_dir='.. / products', calintents='*PHASE*')
```

## 3.7 pipeline.hifa.cli.hifa\_flagdata

**hifa\_flagdata**(vis=None, autocorr=None, shadow=None, tolerance=None, scan=None, scannumber=None, intents=None, edgespw=None, fracspw=None, fracspwfps=None, online=None, partialpol=None, lowtrans=None, mintransrepbw=None, mintransnonrepbw=None, fileonline=None, template=None, filetemplate=None, hm\_tbuff=None, tbuff=None, qa0=None, qa2=None, flagbackup=None, parallel=None) → ResultsList[Results]

Do metadata based flagging of a list of MeasurementSets.

The hifa\_flagdata data performs basic flagging operations on a list of measurements including:

- applying online flags
- applying a flagging template
- partial polarization flagging
- autocorrelation data flagging
- shadowed antenna data flagging
- scan-based flagging by intent or scan number
- edge channel flagging, as needed
- low atmospheric transmission flagging

About the spectral window edge channel flagging:

- For TDM spectral windows, a number of edge channels are always flagged, according to the `fracspw` and `fracspwfps` parameters (the latter operates only on spectral windows with 62, 124, or 248 channels). With the default setting of `fracspw`, the number of channels flagged on each edge is 2, 4, or 8 for 64, 128, or 256-channel spectral windows, respectively.
- For most FDM spectral windows, no edge flagging is done. The only exceptions are ACA spectral windows that encroach too close to the baseband edge. Channels that lie closer to the baseband edge than the following values are flagged: 62.5, 40, 20, 10, and 5 MHz for spectral windows with bandwidths of 1000, 500, 250, 125, and 62.5 MHz, respectively. A warning is generated in the weblog if flagging occurs due to proximity to the baseband edge. By definition, 2000 MHz spectral windows always encroach the baseband edge on both sides of the spectral window, and thus are always flagged on both sides in order to achieve 1875 MHz bandwidth (in effect, they are flagged by 62.5 MHz on each side), and thus no warning is generated.

## Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets defined in the pipeline context.
- **autocorr** -- Flag autocorrelation data.
- **shadow** -- Flag shadowed antennas.
- **tolerance** -- Amount of antenna shadowing tolerated, in meters. A positive number allows antennas to overlap in projection. A negative number forces antennas apart in projection. Zero implies a distance of radius\_1+radius\_2 between antenna centers.
- **scan** -- Flag a list of specified scans.
- **scannumber** -- A string containing a comma delimited list of scans to be flagged.  
Example: `scannumber='3,5,6'`
- **intents** -- A string containing a comma delimited list of intents against which the scans to be flagged are matched.  
Example: `intents='*BANDPASS*''`
- **edgespw** -- Flag the edge spectral window channels.
- **fracspw** -- Fraction of channels to flag at both edges of TDM spectral windows.
- **fracspwfps** -- Fraction of channels to flag at both edges of ACA TDM spectral windows that were created with the earlier (original) implementation of the frequency profile synthesis (FPS) algorithm.
- **online** -- Apply the online flags.
- **partialpol** -- Identify integrations in multi-polarisation data where part of the polarization products are already flagged, and flag the other polarization products in those integrations.
- **lowtrans** -- Flag spectral windows for which a significant fraction of the channels have atmospheric transmission below the threshold (`mintransrepspw`, `mintransnonrepspws`).
- **mintransrepspw** -- This atmospheric transmissivity threshold is used to flag the representative science spectral window when more than 60% of its channels have a transmissivity below this level.
- **mintransnonrepspws** -- This atmospheric transmissivity threshold is used to flag a non-representative science spectral window when more than 60% of its channels have a transmissivity below this level.
- **fileonline** -- File containing the online flags. These are computed by the `h_init` or `hifa_importdata` data tasks. If the online flags files are undefined a name of the form '`msname.flagonline.txt`' is assumed.
- **template** -- Apply flagging templates
- **filitemap** -- The name of a text file that contains the flagging template for RFI, birdies, telluric lines, etc. If the template flags files is undefined a name of the form '`msname.flagtemplate.txt`' is assumed.
- **hm\_tbuff** -- The heuristic for computing the default time interval padding parameter. The options are 'halfint' and 'manual'. In 'halfint' mode tbuff is set to half the maximum of the median integration time of the science and calibrator target observations. The value of 0.048 seconds is subtracted from the lower time limit to accommodate the behavior of the ALMA Control system.
- **tbuff** -- The time in seconds used to pad flagging command time intervals if `hm_tbuff` = 'manual'. The default in manual mode is no flagging.
- **qa0** -- QA0 flags.
- **qa2** -- QA2 flags.
- **flagbackup** -- Back up any pre-existing flags.

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: 'automatic', 'true', 'false', True, False  
Default: **None** (equivalent to **False**)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Do basic flagging on a MeasurementSet:

```
>>> hifa_flagdata()
```

2. Do basic flagging on a MeasurementSet flagging additional scans selected by number as well:

```
>>> hifa_flagdata(scannumber='13,18')
```

## 3.8 pipeline.hifa.cli.hifa\_flagtargets

**hifa\_flagtargets**(vis=None, template=None, filetemplate=None, flagbackup=None, parallel=None) → ResultsList[Results]

Do science target flagging.

The hifa\_flagtargets task performs basic flagging operations on a list of science target MeasurementSets, including:

- applying a flagging template

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets defined in the pipeline context.
- **template** -- Apply flagging templates; defaults to True.
- **filetemplate** -- The name of a text file that contains the flagging template for issues with the science target data etc. If the template flags files is undefined a name of the form 'msname\_flagtargetstemplate.txt' is assumed.
- **flagbackup** -- Back up any pre-existing flags; defaults to False.
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: 'automatic', 'true', 'false', True, False  
Default: **None** (equivalent to **False**)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Do basic flagging on a science target MeasurementSet:

```
>>> hifa_flagtargets()
```

## 3.9 pipeline.hifa.cli.hifa\_fluxcalflag

**hifa\_fluxcalflag**(*vis=None*, *field=None*, *intent=None*, *spw=None*, *threshold=None*, *appendlines=None*, *linesfile=None*, *applyflags=None*) → ResultsList[Results]

Locate and flag line regions in solar system flux calibrators.

Search the built-in solar system flux calibrator line catalog for overlaps with the science spectral windows. Generate a list of line overlap regions and flagging commands.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets defined in the pipeline context.

- **field** -- The list of field names or field ids for which the models are to be set. Defaults to all fields with intent 'AMPLITUDE'.

Example: field='3C279', field='3C279, M82'

- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched. Defaults to all data with amplitude intent.

Example: intent='AMPLITUDE'

- **spw** -- Spectral windows and channels for which bandpasses are computed. Defaults to all science spectral windows.

Example: spw='11,13,15,17'

- **threshold** -- If the fraction of a spectral window occupied by line regions is greater than this threshold value, then flag the entire spectral window.

- **appendlines** -- Append user defined line regions to the line dictionary.

- **linesfile** -- Read in a file containing lines regions and append it to the builtin dictionary. Blank lines and comments beginning with # are skipped. The data is contained in 4 whitespace delimited fields containing the solar system object field name, e.g. 'Callisto', the molecular species name, e.g. '13CO', and the starting and ending frequency in GHz.

- **applyflags** -- Boolean for whether to apply the generated flag commands. (default True)

### Returns

The results object for the pipeline task is returned.

### Examples

1. Locate known lines in any solar system object flux calibrators:

```
>>> hifa_fluxcalflag()
```

## 3.10 pipeline.hifa.cli.hifa\_gaincalsnr

**hifa\_gaincalsnr**(*vis=None*, *field=None*, *intent=None*, *spw=None*, *bwedgefrac=None*, *hm\_nantennas=None*, *maxfracflagged=None*) → ResultsList[Results]

Compute gaincal signal-to-noise ratios per spw.

The gaincal solution signal-to-noise is determined as follows:

- For each data set the list of source(s) to use for the per-scan gaincal solution signal-to-noise estimation is compiled based on the values of the field, intent, and spw parameters.

- Source fluxes are determined for each spw and source combination.
- Fluxes in Jy are derived from the pipeline context.
- Pipeline context fluxes are derived from the online flux calibrator catalog, the ASDM, or the user via the flux.csv file.
- If no fluxes are available the task terminates.
- Atmospheric calibration and observations scans are determined for each spw and source combination.
- If intent is set to 'PHASE' are there are no atmospheric scans associated with the 'PHASE' calibrator, 'TARGET' atmospheric scans will be used instead.
- If atmospheric scans cannot be associated with any of the spw and source combinations the task terminates.
- Science spws are mapped to atmospheric spws for each science spw and source combinations.
- If mappings cannot be determined for any of the spws the task terminates.
- The median Tsyst value for each atmospheric spw and source combination is determined from the SYSCAL table. Medians are computed first by channel, then by antenna, in order to reduce sensitivity to deviant values.
- The science spw parameters, exposure time(s), and integration time(s) are determined.
- The per scan sensitivity and signal-to-noise estimates are computed per science spectral window. Nominal Tsyst and sensitivity values per receiver band provide by the ALMA project are used for this estimate.
- The QA score is based on how many signal-to-noise estimates greater than the requested signal-to-noise ratio can be computed.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['M82A.ms', 'M82B.ms']`

- **field** -- The list of field names of sources to be used for signal to noise estimation. Defaults to all fields with the standard intent.

Example: `field='3C279'`

- **intent** -- A string containing a comma-delimited list of intents against which the selected fields are matched. Defaults to 'PHASE'.

Example: `intent='BANDPASS'`

- **spw** -- The list of spectral windows and channels for which gain solutions are computed. Defaults to all the science spectral windows for which there are both 'intent' and TARGET intents.

Example: `spw='13,15'`

- **bwedgefrac** -- The fraction of the bandwidth edges that is flagged.

Example: `bwedgefrac=0.0`

- **hm\_nantennas** -- The heuristics for determines the number of antennas to use in the signal to noise estimate. The options are 'all' and 'unflagged'. The 'unflagged' option is not currently supported.

Example: `hm_nantennas='unflagged'`

- **maxfracflagged** -- The maximum fraction of an antenna that can be flagged before it is excluded from the signal to noise estimate.

Example: `maxfracflagged=0.80`

### Returns

The results object for the pipeline task is returned.

## Examples

1. Estimate the per scan gaincal solution sensitivities and signal to noise ratios for all the science spectral windows:

```
>>> hifa_gaincalsnr()
```

## 3.11 pipeline.hifa.cli.hifa\_gfluxscale

**hifa\_gfluxscale**(*vis=None*, *reference=None*, *transfer=None*, *refintent=None*, *transintent=None*, *refspwmap=None*, *refile=None*, *phaseupsolint=None*, *solint=None*, *minsnr=None*, *refant=None*, *hm\_resolvedcals=None*, *antenna=None*, *peak\_fraction=None*, *amp\_outlier\_sigma=None*, *parallel=None*) → ResultsList[Results]

Derive flux density scales from standard calibrators.

Derive flux densities for point source transfer calibrators using flux models for reference calibrators.

Flux values are determined by:

- computing phase-up solutions for all the science spectral windows using the calibrator data selected by the **reference** and **refintent** parameters and the **transfer** and **transintent** parameters, with solint and gaintype parameters, and spw mapping or combination as determined in `hifa_spwphaseup`. If no solint is defined, **phaseupsolint** is used (default = 'int').
- computing amplitude only solutions for all the science spectral windows using calibrator data selected with **reference** and **refintent** parameters and the **transfer** and **transintent** parameters.
- examining the amplitude-only solutions for obvious outliers and flagging them in the caltable.
- transferring the flux scale from the reference calibrators to the transfer calibrators using **refspwmap** for windows without data in the reference calibrators.
- inserting the computed flux density values into the MODEL\_DATA column.

Resolved calibrators are handled via antenna selection either automatically (**hm\_resolvedcals** = 'automatic') or manually (**hm\_resolvedcals** = 'manual'). In the former case, antennas closer to the reference antenna than the uv distance where visibilities fall to **peak\_fraction** of the peak are used. In manual mode, the antennas specified in **antenna** are used.

Note that the flux corrected calibration table computed internally is not currently used in later pipeline apply calibration steps because the relevant calibrator flux densities have been set in the MODEL\_DATA column.

Note that for very low SNR, noise bias already positively skews the amplitudes. The time solint could be greater for either **transintent** or **refintent**, which has a compounding effect: if the phase stability is insufficient over that longer solint, an optimal phaseup will not be able to be calculated, and this also results in an artificial increase in amplitudes because the amplitude gains compensate for the uncorrected decorrelation.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.  
Example: `['M32A.ms', 'M32B.ms']`
- **reference** -- A string containing a comma delimited list of field names defining the reference calibrators. Defaults to names of fields with intents in **refintent**.  
Example: `reference='M82,3C273'`
- **transfer** -- A string containing a comma delimited list of field names defining the transfer calibrators. Defaults to names of fields with intents in **transintent**.  
Example: `transfer='J1328+041,J1206+30'`
- **refintent** -- A string containing a comma delimited list of intents used to select the reference calibrators. Defaults to 'AMPLITUDE'.

Example: `refintent=' ', r`efintent='AMPLITUDE'`

- **transintent** -- A string containing a comma delimited list of intents defining the transfer calibrators. Defaults to 'PHASE,BANDPASS,CHECK,DIFFGAINREF,DIFFGAINSRC,POLARIZATION,POLANGLE,POLLEAKAGE'.

Example: `transintent=' ', transintent='PHASE,BANDPASS'`

- **refspwmap** -- Vector of spectral window ids enabling scaling across spectral windows. Defaults to no scaling.

Example: `refspwmap=[1,1,3,3]` - (4 spws, reference fields in 1 and 3, transfer fields in 0,1,2,3)

- **reffile** -- Path to a file containing flux densities for calibrators. Setjy will be run for any that have both reference and transfer intents. Values given in this file will take precedence over MODEL column values set by previous tasks. By default, the path is set to the CSV file created by hifa\_importdata, consisting of catalogue fluxes extracted from the ASDM and / or edited by the user.

Example: `reffile=' ', reffile='working/flux.csv'`

- **phaseupsolint** -- Time solution intervals in CASA syntax for the phase solution.

Example: `phaseupsolint='inf', phaseupsolint='int', phaseupsolint='100sec'`

- **solint** -- Time solution intervals in CASA syntax for the amplitude solution.

Example: `solint='inf', solint='int', solint='100sec'`

- **minsnr** -- Minimum signal-to-noise ratio for gain calibration solutions.

Example: `minsnr=1.5, minsnr=0.0`

- **refant** -- A string specifying the reference antenna(s). By default, this is read from the context.

Example: `refant='DV05'`

- **hm\_resolvedcals** -- Heuristics method for handling resolved calibrators. The options are 'automatic' and 'manual'. In automatic mode, antennas closer to the reference antenna than the uv distance where visibilities fall to `peak_fraction` of the peak are used. In manual mode, the antennas specified in `antenna` are used.

- **antenna** -- A comma delimited string specifying the antenna names or ids to be used for the fluxscale determination. Used in `hm_resolvedcals='manual'` mode.

Example: `antenna='DV16,DV07,DA12,DA08'`

- **peak\_fraction** -- The limiting UV distance from the reference antenna for antennas to be included in the flux calibration. Defined as the point where the calibrator visibilities have fallen to `peak_fraction` of the peak value.

- **amp\_outlier\_sigma** -- Sigma threshold used to identify outliers in the amplitude caltable. Default: 50.0.

Example: `amp_outlier_sigma=30.0`

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: `'automatic', 'true', 'false', True, False`

Default: `None` (equivalent to `False`)

## Returns

The results object for the pipeline task is returned.

## Examples

1. Compute flux values for the phase calibrator using model data from the amplitude calibrator:

```
>>> hifa_gfluxscale()
```

## 3.12 pipeline.hifa.cli.hifa\_gfluxscaleflag

**hifa\_gfluxscaleflag**(vis=None, intent=None, phaseupsolint=None, solint=None, minsnr=None, refant=None, antnegsig=None, antpossig=None, tmantint=None, tmint=None, tmb=None, antblnegsig=None, antblpossig=None, relaxed\_factor=None, niter=None, parallel=None) → ResultsList[Results]

Flag the flux, diffgain, phase calibrators and check source.

This task computes the flagging heuristics on the flux, diffgain, and phase calibrators and the check source, by calling hif\_correctedampflag which looks for outlier visibility points by statistically examining the scalar difference of corrected amplitudes minus model amplitudes, and flags those outliers. The philosophy is that only outlier data points that have remained outliers after calibration will be flagged. The heuristic works equally well on resolved calibrators and point sources because it is not performing a vector difference, and thus is not sensitive to nulls in the flux density vs. uvdistance domain. Note that the phase of the data is not assessed.

In further detail, the workflow is as follows: a snapshot of the flagging state is preserved at the start, a preliminary phase and amplitude gaincal solution is solved and applied, the flagging heuristics are run and any outliers are marked for flagging, the flagging state is restored from the snapshot. If any outliers were found, then these are flagged. Plots are generated at two points in this workflow: after preliminary phase and amplitude calibration but before flagging heuristics are run, and after flagging heuristics have been run and applied. If no points were flagged, the 'after' plots are not generated or displayed. The score for this stage is the standard data flagging score, which depends on the fraction of data flagged.

The preliminary phase solutions use the mapping/combine and gaintype options as established in hifa\_spwphaseup.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['M51.ms']`

- **intent** -- A string containing a comma delimited list of intents against which the selected fields are matched. If undefined (default), it will select all data with the AMPLITUDE, PHASE, and CHECK intents, except for one case: if one of the AMPLITUDE intent fields was also used for BANDPASS, then this task will select only data with PHASE and CHECK intents.

Example: `intent='*PHASE*'`

- **phaseupsolint** -- The phase correction solution interval in CASA syntax.

Example: `phaseupsolint='300s'`

- **solint** -- Time and channel solution intervals in CASA syntax.

Example: `solint='inf,10ch', solint='inf'`

- **minsnr** -- Solutions below this SNR are rejected.

- **refant** -- Reference antenna names. Defaults to the value(s) stored in the pipeline context. If undefined in the pipeline context defaults to the CASA reference antenna naming scheme.

Example: `refant='DV01', refant='DV06,DV07'`

- **antnegsig** -- Lower sigma threshold for identifying outliers as a result of bad antennas within individual timestamps.

Example: `antnegsig=4.0`

- **antpossig** -- Upper sigma threshold for identifying outliers as a result of bad antennas within individual timestamps.

Example: `antpossig=4.6`

- **tmantint** -- Threshold for maximum fraction of timestamps that are allowed to contain outliers.  
Example: `tmantint=0.063`
- **tmint** -- Threshold for maximum fraction of "outlier timestamps" over "total timestamps" that a baseline may be a part of.  
Example: `tmint=0.085`
- **tmb1** -- Initial threshold for maximum fraction of "bad baselines" over "all baselines" that an antenna may be a part of.  
Example: `tmb1=0.175`
- **antblnegsig** -- Lower sigma threshold for identifying outliers as a result of "bad baselines" and/or "bad antennas" within baselines, across all timestamps.  
Example: `antblnegsig=3.4`
- **antblpossig** -- Threshold for identifying outliers as a result of "bad baselines" and/or "bad antennas" within baselines, across all timestamps.  
Example: `antblpossig=3.2`
- **relaxed\_factor** -- Relaxed value to set the threshold scaling factor to under certain conditions (see task description).  
Example: `relaxed_factor=2.0`
- **niter** -- Maximum number of times to iterate on evaluation of flagging heuristics. If an iteration results in no new flags, then subsequent iterations are skipped.  
Example: `niter=2`
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`  
Default: `None` (equivalent to `False`)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. run with recommended settings to create flux scale calibration with flagging using recommended thresholds:

```
>>> hifa_gfluxscaleflag()
```

## 3.13 pipeline.hifa.cli.hifa\_imageprecheck

**hifa\_imageprecheck**(vis=None, desired\_angular\_resolution=None, calcsb=None, parallel=None) → Results

Calculates the best Briggs robust parameter to achieve sensitivity and angular resolution goals.

In this task, the representative source and the spw containing the representative frequency selected by the PI in the OT are used to calculate the synthesized beam and to make sensitivity estimates for the aggregate bandwidth and representative bandwidth for several values of the Briggs robust parameter. This information is reported in a table in the weblog. If no representative target/frequency information is available, it defaults to the first target and center of first spw in the data (e.g. pre-Cycle 5 data does not have this information available). The best Briggs robust parameter to achieve the PI's desired angular resolution is chosen automatically. See the User's guide for further details.

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifa\_importdata task. ": use all MeasurementSets in the context  
Examples: 'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']
- **desired\_angular\_resolution** -- User specified angular resolution goal string. When this parameter is set, uv tapering may be performed. ": automatic from performance parameters (default).  
Example: '1.0arcsec'
- **calcsb** -- Force (re-)calculation of sensitivities and beams; defaults to False
- **parallel** -- Use the CASA imager parallel processing when possible.  
Options: 'automatic', 'true', 'false', True, False  
Default: 'automatic'

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. run with recommended settings to perform checks prior to imaging:

```
>>> hifa_imageprecheck()
```

2. run to perform checks prior to imaging and force the re-calculation of sensitivities and beams:

```
>>> hifa_imageprecheck(calcsb=True)
```

## 3.14 pipeline.hifa.cli.hifa\_importdata

```
hifa_importdata(vis: list[str] | None = None, session: str | None = None, asis: str | None = None, process_caldevice: bool | None = None, overwrite: bool | None = None, nocopy: bool | None = None, bdfflags: bool | None = None, datacolumns: dict[str, str] | None = None, lazy: bool | None = None, dbservice: bool | None = None, ocorr_mode: str | None = None, createmms: str | None = None, minparang: float | None = None, parallel: bool | None = None) → ResultsList[Results]
```

Imports data into the interferometry pipeline.

The hifa\_importdata task loads the specified visibility data into the pipeline context unpacking and / or converting it as necessary.

If the **overwrite** input parameter is set to False and the task is asked to convert an input ASDM input to an MS, then when the output MS already exists in the output directory, the importasdm conversion step is skipped, and the existing MS will be imported instead.

**Parameters**

- **vis** -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSes, or tar files of MSes. If ASDM files are specified, they will be converted to MS format.  
Example: vis=['X227.ms', 'asdms.tar.gz']
- **session** -- List of session names, one for each visibility dataset, used to group the MSes into sessions.  
Example: session=['session\_1', 'session\_2']
- **asis** -- Creates verbatim copies of the ASDM tables in the output MS. The value given to this option must be a list of table names separated by space characters.
- **process\_caldevice** -- Import the caldevice table from the ASDM.

- **overwrite** -- Overwrite existing files on import; defaults to False. When converting ASDM to MS, if overwrite=False and the MS already exists in the output directory, then this existing MS dataset will be used instead.

Example: `overwrite=True`

- **nocopy** -- Disable copying of MS to working directory; defaults to False.

Example: `nocopy=True`

- **bdfflags** -- Apply BDF flags on import.

- **datacolumns** -- Dictionary defining the data types of existing columns. The format is:

```
{'data': 'data type 1'}
```

or

```
{'data': 'data type 1', 'corrected': 'data type 2'}.
```

For ASDMs the data type can only be RAW and one can only specify it for the data column. For MSes one can define two different data types for the DATA and CORRECTED\_DATA columns and they can be any of the known data types (RAW, REGCAL\_CONTLINE\_ALL, REGCAL\_CONTLINE\_SCIENCE, SELFCAL\_CONTLINE\_SCIENCE, REGCAL\_LINE\_SCIENCE, SELFCAL\_LINE\_SCIENCE, BASELINED, ATMCORR). The intent selection strings \_ALL or \_SCIENCE can be skipped. In that case the task determines this automatically by inspecting the existing intents in the dataset. Usually, a single datacolumns dictionary is used for all datasets. If necessary, one can define a list of dictionaries, one for each EB, with different setups per EB. If no types are specified, {'data':'raw','corrected':'regcal\_contline'} or {'data':'raw'} will be assumed, depending on whether the corrected column exists or not.

- **lazy** -- Use the lazy filler import.

- **dbservice** -- Use the online flux catalog.

- **ocorr\_mode** -- ALMA default set to ca.

- **createmms** -- Create an MMS.

- **minparang** -- Minimum required parallactic angle range for polarisation calibrator, in degrees. The default of 0.0 is used for non-polarisation processing.

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`

Default: `None` (equivalent to `False`)

## Returns

The results object for the pipeline task is returned.

## Examples

- Load an ASDM list in the .. rawData subdirectory into the context:

```
>>> hifa_importdata(vis=['.. rawData/uid__A002_X30a93d_X43e', '.. rawData/uid_A002_x30a93d_X44e  
  '])
```

- Load an MS in the current directory into the context:

```
>>> hifa_importdata(vis=['uid__A002_X30a93d_X43e.ms'])
```

- Load a tarred ASDM in .. rawData into the context:

```
>>> hifa_importdata(vis=['./rawdata/uid__A002_X30a93d_X43e.tar.gz'])
```

4. Import a list of MeasurementSets:

```
>>> myvislist = ['uid__A002_X30a93d_X43e.ms', 'uid_A002_x30a93d_X44e.ms']
>>> hifa_importdata(vis=myvislist)
```

5. Run with explicit setting of data column types:

```
>>> hifa_importdata(vis=['uid__A002_X30a93d_X43e_targets.ms'], datacolumns={'data': 'regcal_'
->contline'})
>>> hifa_importdata(vis=['uid__A002_X30a93d_X43e_targets_line.ms'], datacolumns={'data': 'regcal_'
->line', 'corrected': 'selfcal_line'})
```

## 3.15 pipeline.hifa.cli.hifa\_lock\_refant

**hifa\_lock\_refant**(*vis=None*) → ResultsList[Results]

Lock reference antenna list.

*hifa\_lock\_refant* marks the reference antenna list as "locked" for specified MeasurementSets, preventing modification of the refant list by subsequent tasks.

After executing *hifa\_lock\_refant*, all subsequent gaincal calls will by default be executed with *refantmode='strict'*.

The refant list can be unlocked with the *hifa\_unlock\_refant* task.

### Parameters

- **vis** -- List of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['ngc5921.ms']`

### Returns

The results object for the pipeline task is returned.

### Examples

1. Lock the refant list for all MSes in pipeline context:

```
>>> hifa_lock_refant()
```

## 3.16 pipeline.hifa.cli.hifa\_polcal

**hifa\_polcal**(*vis=None, minpacov=None, solint\_chavg=None, vs\_stats=None, vs\_thresh=None*) → PolcalResults

Derive instrumental polarization calibration for ALMA.

Derive the instrumental polarization calibrations for ALMA using the polarization calibrators.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `['M32A.ms', 'M32B.ms']`

- **minpacov** -- Minimum Parallactic Angle Coverage (degrees) required for Q, U estimation in task polfromgain. This enables avoiding pathological cases of antennas with insufficient parallactic angle coverage which can yield spurious source polarization solutions or cause polfromgain to fail to find any solutions.

Default: 30.0

- **solint\_chavg** -- Channel averaging to include in solint for gaincal steps producing cross-hand delay, cross-hand phase, and leakage (D-terms) solutions.

Default: '5MHz'

- **vs\_stats** -- List of visstat statistics to use for diagnostic comparison between the concatenated session MS and individual MSes in that session after applying polarization calibration tables.

Default: ['min','max','mean']

- **vs\_thresh** -- Threshold to use in diagnostic comparison of visstat statistics; relative differences larger than this threshold are reported in the CASA log.

Default: 1e-3

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Compute the polarization calibrations:

```
>>> hifa_polcal()
```

## 3.17 pipeline.hifa.cli.hifa\_polcalfag

**hifa\_polcalfag**(vis=None) → ResultsList[Results]

Flag polarization calibrators.

This task flags corrected visibility outliers in the polarization calibrator data using the hif\_correctedampflag heuristics.

#### Parameters

**vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifa\_importdata task. ":" use all MeasurementSets in the context

Examples: 'ngc5921.ms', ['ngc5921a.ms', ngc5921b.ms', 'ngc5921c.ms']

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Run with recommended settings to flag visibility outliers in the polarization calibrator data:

```
>>> hifa_polcalfag()
```

## 3.18 pipeline.hifa.cli.hifa\_renorm

---

**hifa\_renorm**(*vis=None*, *createcaltable=None*, *threshold=None*, *spw=None*, *excludechan=None*, *atm\_auto\_exclude=None*, *bwthreshspw=None*, *parallel=None*) → ResultsList[Results]

ALMA renormalization.

This task makes an assessment, and optionally applies a correction, to data suffering from incorrect amplitude normalization caused by bright astronomical lines detected in the autocorrelations of some target sources.

For a full description of the effects of bright emission lines and the correction heuristics used in this task, please see the Pipeline User Guide.

### Parameters

- **vis** -- List of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['ngc5921.ms']`

- **createcaltable** -- Boolean to select whether to create the renormalization correction cal table (True), or only run the assessment (False, default).

Example: `createcaltable=True`

- **threshold** -- Apply correction if max correction is above this threshold value and `apply= True`. Default is 1.02 (i.e. 2%).

Example: `threshold=1.02`

- **spw** -- The list of real (not virtual - i.e. the actual spwIDs in the MS) spectral windows to evaluate. Set to `spw="` by default, which means the task will select all relevant (science FDM) spectral windows. Note that for data with multiple MSs, a list with the correct spectral window selection for each MS can be provided.

Examples:

– `spw='11,13,15,17'`

– `spw=['11, 13, 15, 17', '5,7,11,13']`

- **excludechan** -- Channels to exclude in either channel or frequency space (TOPO, GHz), specifying the real (not virtual) spectral window per selection. Note that for data with multiple MSs, a list of dictionaries with the correct selection for each MS can be provided.

Examples:

– `excludechan={'22':'100~150;800~850', '24':'100~200'}`

– `excludechan={'22':'230.1GHz~230.2GHz'}`

– `excludechan=[{'22':'100~150'}, {'15':'100~150'}]`

- **atm\_auto\_exclude** -- Automatically find and exclude regions with atmospheric features. Default is False

- **bwthreshspw** -- Bandwidth beyond which a SPW is split into chunks to fit separately. The default value for all SPWs is 120e6, and this parameter allows one to override it for specific SPWs, due to needing potentially various 'nsegments' when EBs have very different SPW bandwidths.

Example: `bwthreshspw={'16: 64e6, '22: 64e6'}`

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: `'automatic', 'true', 'false', True, False`

Default: `None` (equivalent to `automatic`)

### Returns

The results object for the pipeline task is returned.

## Examples

1. Run with recommended settings to assess the need for an ALMA amplitude renormalization correction.

```
>>> hifa_renorm()
```

2. Run to assess the necessary ALMA amplitude renormalization correction, and apply this correction if it exceeds a threshold of 3% (1.03).

```
>>> hifa_renorm(createcaltable=True, threshold=1.03)
```

## 3.19 pipeline.hifa.cli.hifa\_restoredata

**hifa\_restoredata**(*vis=None*, *session=None*, *products\_dir=None*, *copytoraw=None*, *rawdata\_dir=None*, *lazy=None*, *bdfflags=None*, *ocorr\_mode=None*, *asis=None*) → Results

Restore flagged and calibration interferometry data from a pipeline run.

The *hifa\_restoredata* task restores flagged and calibrated MeasurementSets from archived ASDMs and pipeline flagging and calibration data products.

*hifa\_restoredata* assumes that the ASDMs to be restored are present in the directory specified by the *rawdata\_dir* (default: './rawdata').

By default (*copytoraw* = True), *hifa\_restoredata* assumes that for each ASDM in the input list, the corresponding pipeline flagging and calibration data products (in the format produced by the *hifa\_exportdata* task) are present in the directory specified by *products\_dir* (default: './products'). At the start of the task, these products are copied from the *products\_dir* to the *rawdata\_dir*.

If *copytoraw* = False, *hifa\_restoredata* assumes that these products are to be found in *rawdata\_dir* along with the ASDMs.

The expected flagging and calibration products (for each ASDM) include:

- a compressed tar file of the final flagversions file, e.g. uid\_\_A002\_X30a93d\_X43e.ms.flagversions.tar.gz
- a text file containing the applycal instructions, e.g. uid\_\_A002\_X30a93d\_X43e.ms.calapply.txt
- a compressed tar file containing the caltables for the parent session, e.g. uid\_\_A001\_X74\_X29.session\_3.caltables.tar.gz

*hifa\_restoredata* performs the following operations:

- imports the ASDM(s)
- removes the default MS.flagversions directory created by the filler
- restores the final MS.flagversions directory stored by the pipeline
- restores the final set of pipeline flags to the MS
- restores the final calibration state of the MS
- restores the final calibration tables for each MS
- applies the calibration tables to each MS

When importing the ASDM and converting it to a MeasurementSet (MS), if the output MS already exists in the output directory, then the importasdm conversion step is skipped, and instead the existing MS will be imported.

### Parameters

- **vis** -- List of raw visibility data files to be restored. Assumed to be in the directory specified by *rawdata\_dir*.

Example: `vis=['uid__A002_X30a93d_X43e']`

- **session** -- List of sessions one per visibility file.

Example: `session=['session_3']`

- **products\_dir** -- Name of the data products directory to copy calibration products from. Default: `'./products'`. The parameter is effective only when `copytoraw` = True. When `copytoraw` = False, calibration products in `rawdata_dir` will be used.

Example: `products_dir='myproductspath'`

- **copytoraw** -- Copy calibration and flagging tables from `products_dir` to `rawdata_dir` directory.

Default: `True`.

Example: `copytoraw=False`

- **rawdata\_dir** -- Name of the raw data directory.

Default: `'../rawdata'`.

Example: `rawdata_dir='myrawdatopath'`

- **lazy** -- Use the lazy filler option.

Default: `False`.

Example: `lazy=True`

- **bdfflags** -- Set the BDF flags.

Default: True.

Example: `bdfflags=False`

- **ocorr\_mode** -- Set ocorr\_mode.

Default: `'ca'`.

Example: `ocorr_mode='ca'`

- **asis** -- Creates verbatim copies of the ASDM tables in the output MS. The value given to this option must be a string containing a list of table names separated by whitespace characters.

Default: `'SBSummary ExecBlock Antenna Annotation Station Receiver Source CalAtmosphere CalWVR CalPointing'`.

Example: `asis='Source Receiver'`

## Returns

The results object for the pipeline task is returned.

## Examples

1. Restore the pipeline results for a single ASDM in a single session:

```
>>> hifa_restoredata(vis=['uid__A002_X30a93d_X43e'], session=['session_1'], ocorr_mode='ca')
```

## 3.20 pipeline.hifa.cli.hifa\_session\_refant

**hifa\_session\_refant**(*vis=None*, *phase\_threshold=None*) → Results

Select best reference antenna for session(s).

This task re-evaluates the reference antenna lists from all MeasurementSets within a session and combines these to select a single common reference antenna (per session) that is to be used by any subsequent pipeline stages.

## Parameters

- **vis** -- List of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.  
Example: `vis=['ngc5921.ms']`
- **phase\_threshold** -- Threshold (in degrees) used to identify absolute phase solution outliers in caltables.  
Example: `phase_threshold=0.005`

## Returns

The results object for the pipeline task is returned.

## Examples

1. Compute a single common reference antenna per session:

```
>>> hifa_session_refant()
```

## 3.21 pipeline.hifa.cli.hifa\_spwphaseup

**hifa\_spwphaseup**(*vis=None*, *caltable=None*, *field=None*, *intent=None*, *spw=None*, *hm\_spwmapmode=None*, *maxnarrowbw=None*, *minfracmaxbw=None*, *samebb=None*, *phasesnr=None*, *intphasesnr=None*, *intphasesnrmin=None*, *phaseupmaxsolint=None*, *bwedgefrac=None*, *hm\_nantennas=None*, *maxfracflagged=None*, *combine=None*, *refant=None*, *minblperant=None*, *minsnr=None*, *unregister\_existing=None*) → ResultsList[Results]

Compute phase calibration spw map and per spw phase offsets.

*hifa\_spwphaseup* computes the spw map for phase calibration, and derives phase offsets as a function of spectral window using high signal-to-noise calibration observations. Previous calibrations are applied on the fly.

*hifa\_spwphaseup* performs two functions:

- Determines the spectral window mapping or combination mode, gaintype and solint, for each independent bandpass, amplitude, diffgain, phase and check source, to use when solving the phaseup (phase as a function of time) in subsequent stages (mapping mode and gaintype in *hifa\_gfluxscaleflag*, all parameters for *hifa\_gfluxscale* and *hifa\_timegaincal*), and when applying those solutions to targets.
- Computes the per-spectral-window phase offset table that will be applied to the data to remove mean phase differences between the spectral windows.

If `hm_spwmapmode='auto'`, then the spectral window map is computed for each SpectralSpec and each calibrator source, using the following algorithm:

- Estimate the per-spectral-window (spw) signal-to-noise ratio based on catalog flux densities, Tsys, number of antennas, and integration scan time. These estimates are shown in the weblog.
- Compute the per-spw signal-to-noise for each above mentioned intent based on an temporary phaseup gain calibration using solint `inf` for the PHASE and CHECK intents, and `int` otherwise. These computed signal-to-noise values override the use of the estimated values unless none are found.
- If SNR calculation fails, then subsequent heuristics are skipped, warnings printed, and mapping falls back to narrow-to-wide.
- If the signal-to-noise of all spws is greater than `phasesnr` for a PHASE or CHECK intent, or greater than `intphasesnr` for the other intents, then if SNR is high enough to keep `solint='int'`, then no mapping is used (each spw is used to calibrate itself). If the calculated solint is `>'int'` then mapping and combine are attempted, to favor a short solint over keeping the spws independent.
- If the signal-to-noise of only some spws are greater than the value of `phasesnr` for the PHASE or CHECK intent, or `intphasesnr` for the other intents, then each lower-SNR spw is mapped to the highest SNR spw in the same SpectralSpec.

- If all spws have low SNR (or all spws in a given SpectraSpec for multi-SpectralSpec observations), then spws are combined.
- The time solint is calculated and stored for phaseup in subsequent stages. If `hm_spwmapmode` is not '`combine`' then there is at least one high signal-to-noise spw, then by definition that has a solint of '`int`'. For `hm_spwmapmode='combine'`, the computed signal-to-noise is used in conjunction with `phasesnr` for the phase and check intents, or `intphasesnr` for the other intents. First the gaintype is changed from 'G' to 'T' (combine polarization) and thereafter solint is increased from '`int`' up to the limits of 1/2 scan for the phase or check intent, or to the input `maxphaseupsolint` for the other intents
- If `hm_spwmapmode` is '`combine`' and no SNRs are found, the PHASE and CHECK intents will default to a solint of 1/4 scan.
- If the intent is AMPLITUDE, and gaintype was changed from 'G' to 'T', the signal-to-noise required to be met before increasing solint above '`int`' is reduced to `intphasesnrmin`.

If `hm_spwmapmode='combine'`, `hifa_spwphaseup` maps all the science windows to a single science spectral window. For example, if the list of science spectral windows is [9, 11, 13, 15] then all the science spectral windows in the data will be combined and mapped to the science window 9 in the combined phase vs time calibration table.

If `hm_spwmapmode='simple'`, a mapping from narrow science to wider science spectral windows is computed using the following algorithm:

- Construct a list of the bandwidths of all the science spectral windows.
- Determine the maximum bandwidth in this list as `maxbandwidth`.
- For each science spectral window with bandwidth less than `maxbandwidth`, construct a list of spectral windows with bandwidths greater than `minfracmaxbw * maxbandwidth`, then select the spectral window in this list whose band center most closely matches the band center of the narrow spectral window, and preferentially match within the same baseband if `samebb=True`.

If `hm_spwmapmode='default'`, the spw mapping is assumed to be one to one.

After determining the combine and mapping parameters and time solints, phase offsets per spectral window are determined by computing a phase only gain calibration on the selected data, normally the high signal-to-noise bandpass calibrator observations, using the solution interval 'inf'.

At the end of the task the spectral window map, solint and gaintype along with the phase offset calibration table(s) in the pipeline are stored in the context for use by later tasks.

Finally, the SNR of the calibration solutions are inspected and if the median value on a per-spw basis does not reach specific thresholds, a warning is issued with a reduced QA score. For PHASE intent, blue, yellow, and red QA result if the achieved SNR is less than 0.75, 0.5, and 0.33 times `phasesnr`, respectively. For BANDPASS, AMPLITUDE, and DIFFGAIN intent, QA messages are based on 0.75, 0.5, and 0.33 times `intphasesnr` (unless `intphasesnrmin` was used for BANDPASS or AMPLITUDE). Finally, for CHECK intent the QA score is always blue, but scales depending on achieved SNR relative to `phasesnr`.

## Parameters

- **`vis`** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.  
Example: `vis=['M82A.ms', 'M82B.ms']`
- **`caltable`** -- The list of output calibration tables. Defaults to the standard pipeline naming convention.  
Example: `caltable=['M82.gcal', 'M82B.gcal']`
- **`field`** -- The list of field names or field ids for which phase offset solutions are to be computed. Defaults to all fields with the default intent.  
Example: `field='3C279', field='3C279, M82'`
- **`intent`** -- A string containing a comma delimited list of intents against which the selected fields are matched. Defaults to the BANDPASS observations.

Example: `intent='PHASE'`

- **spw** -- The list of spectral windows and channels for which gain solutions are computed. Defaults to all the science spectral windows.

Example: `spw='13,15'`

- **hm\_spwmapmode** -- The spectral window mapping mode. The options are: 'auto', 'combine', 'simple', and 'default'. In 'auto' mode hifa\_spwphaseup estimates the SNR of the phase calibrator observations and uses these estimates to choose between 'combine' mode (low SNR) and 'default' mode (high SNR). In combine mode all spectral windows are combined and mapped to one spectral window. In 'simple' mode narrow spectral windows are mapped to wider ones using an algorithm defined by 'maxnarrowbw', 'minfracmaxbw', and 'samebb'. In 'default' mode the spectral window map defaults to the standard one to one mapping.

Example: `` hm\_spwmapmode='combine' ``

- **maxnarrowbw** -- The maximum bandwidth defining narrow spectral windows. Values must be in CASA compatible frequency units.

Example: `maxnarrowbw=' '`

- **minfracmaxbw** -- The minimum fraction of the maximum bandwidth in the set of spws to use for matching.

Example: `minfracmaxbw=0.75`

- **samebb** -- Match within the same baseband if possible.

Example: `samebb=False`

- **phasesnr** -- The required phase gaincal solution signal-to-noise.

Example: `phasesnr=20.0`

- **intphasesnr** -- The required solint='int' phase gaincal solution signal-to-noise.

Example: `intphasesnr=4.0`

- **intphasesnrmin** -- The required solint='int' phase gaincal solution signal-to-noise for fields that cover the AMPLITUDE calibrator intent.

Example: `intphasesnrmin=3.0`

- **phaseupmaxsolint** -- Maximum phase correction solution interval (in seconds) allowed in very low-SNR cases. Used only when `hm_spwmapmode` = 'auto' or 'combine'.

Example: `phaseupmaxsolint=60.0`

- **bwedgefrac** -- The fraction of the bandwidth edges that is flagged.

Example: `bwedgefrac=0.0`

- **hm\_nantennas** -- The heuristics for determines the number of antennas to use in the signal-to-noise estimate. The options are 'all' and 'unflagged'. The 'unflagged' option is not currently supported.

Example: `hm_nantennas='unflagged'`

- **maxfracflagged** -- The maximum fraction of an antenna that can be flagged before it is excluded from the signal-to-noise estimate.

Example: `maxfracflagged=0.80`

- **combine** -- Data axes to combine for solving. Options are '', 'scan', 'spw', 'field' or any comma-separated combination.

Example: `combine=''`

- **refant** -- Reference antenna name(s) in priority order. Defaults to most recent values set in the pipeline context. If no reference antenna is defined in the pipeline context the CASA defaults are used.

Example: `refant='DV01', refant='DV05,DV07'`

- **minblperant** -- Minimum number of baselines required per antenna for each solve. Antennas with fewer baselines are excluded from solutions.

Example: `minblperant=2`

- **minsnr** -- Solutions below this SNR are rejected.

- **unregister\_existing** -- Unregister previous spwphaseup calibrations from the pipeline context before registering the new calibrations from this task.

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Compute the default spectral window map and the per spectral window phase offsets:

```
>>> hifa_spwphaseup()
```

2. Compute the default spectral window map and the per spectral window phase offsets set the spectral window mapping mode to 'combine':

```
>>> hifa_spwphaseup(hm_spwmapmode='combine')
```

## 3.22 pipeline.hifa.cli.hifa\_targetflag

**hifa\_targetflag**(vis=None, parallel=None) → ResultsList[Results]

Flag target source outliers.

This task flags obvious outliers in the target source data. The calibration tables and flags accumulated in the cal library up to this point are pre-applied, then hif\_correctedampflag is called for just the TARGET intent. Any resulting flags are applied and the calibration library is restored to the state before calling this task.

Because science targets are generally not point sources, the flagging algorithm needs to be more clever than for point source calibrators. The algorithm identifies outliers by examining statistics within successive overlapping radial uv bins, allowing it to adapt to an arbitrary uv structure. Outliers must appear to be a potential outlier in two bins in order to be declared an outlier. To further avoid overflagging of good data, only the highest threshold levels are used (+12/-13 sigma). This stage does not add significant processing time, particularly in making the plots. So to save time, the amp vs. time plots are created only if flags are generated, and the amp vs. uv distance plots are made for only those spws that generated flags. Also, to avoid confusion in mosaics and single field surveys, the amp vs. uv distance plots only show field IDs with new flags.

#### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifa\_importdata task. ": use all MeasurementSets in the context

Examples: 'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: 'automatic', 'true', 'false', True, False

Default: `None` (equivalent to `False`)

#### Returns

The results object for the pipeline task is returned.

## Examples

- Run with recommended settings to flag outliers in science target(s):

```
>>> hifa_targetflag()
```

## 3.23 pipeline.hifa.cli.hifa\_timegaincal

**hifa\_timegaincal**(vis=None, calamptable=None, calphasetable=None, offsetstable=None, targetphasetable=None, amptable=None, field=None, spw=None, antenna=None, calsolint=None, targetsolint=None, refant=None, refantmode=None, solnorm=None, minblperant=None, calminsnr=None, targetminsnr=None, smodel=None, parallel=None) → ResultsList[GaincalResults]

Determine temporal gains from calibrator observations.

The time-dependent complex gains for each antenna/spw are determined from the raw data (DATA column) divided by the model (MODEL column), for the specified fields. The gains are computed according to the spw mapping/combination, solint, and gaintype as determined in hifa\_spwphaseup.

Previous calibrations are applied on the fly.

The process to solve for the various complex gains follows:

- Phase solutions are produced for all intents (excluding the CHECK intent) using the spw mapping/combination and gaintype determined in hifa\_spwphaseup, and using solint = 'inf' (per scan). The solutions only registered for the PHASE intent to transfer to itself (PHASE intent) and the TARGET (and CHECK) intent(s) in hif\_applycal.
- Phase (phase-up) solutions are produced for all intents using the spw mapping/ combination, solint (typically 'int' if not low SNR data) and gaintype as determined in hifa\_spwphaseup. The solutions are used for (1) on-the-fly application as to solve the subsequent amplitude gains, (2) final phase correction in hif\_applycal of the BANDPASS, AMPLITUDE, DIFFGAIN, POLARIZATION intents (i.e. after hif\_applycal, those intents are self-calibrated but PHASE and CHECK are not.) These short-solint phases are also shown as **diagnostic** plots.
- Amplitude solutions are produced for all calibrator intents with the above phase-up solutions pre-applied. The time solint is 'inf' (scan), so solutions are found for each scan and spw. The solutions are registered for AMP gain correction of all intents to themselves, and the PHASE intent to the TARGET and CHECK intent(s). Note: for band-to-band observations, there are no 'high frequency' observations of the PHASE intent, and the full AMP gains are transferred from the AMPLITUDE intent to the TARGET and CHECK intent(s).
- Short term **diagnostic** amplitude solutions are produced for all calibrator intents using the same short solint as that used for the PHASE intent phase-up (typically 'int' except for low SNR cases). These solutions are plotted but not applied.
- Diagnostic phase offsets solutions are produced with solint='inf' for the BANDPASS and PHASE intent, first solving and preapplying the phase as a function of time with combine='spw' (to **zero** the phases), and then then solving phase(time) per spw. Note: by definition the BANDPASS phase will be exactly zero. The phase solutions for the PHASE intent are used by QA heuristics to identify jumps and drifts of the spw-spw offsets as a function of time, but if the SNR is very low, such offsets will not be able to detected.

Good candidate reference antennas were determined using the *hif\_refant* task. During all solutions for standard observing modes, the reference antenna can change flexibly. For polarization observations a good, un-flagged common reference antenna is found and locked in time. For band-to-band observations, all solutions are also made strictly enforcing the use of the selected reference antenna.

Previous calibrations that have been stored in the pipeline context are applied on the fly.

### Parameters

- vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['M82A.ms', 'M82B.ms']`

- **calamptable** -- The list of output diagnostic calibration amplitude tables for the calibration targets. Defaults to the standard pipeline naming convention.

Example: `calamptable=['M82.gacal', 'M82B.gacal']`

- **calphasetable** -- The list of output calibration phase tables for the calibration targets. Defaults to the standard pipeline naming convention.

Example: `calphasetable=['M82.gpcal', 'M82B.gpcal']`

- **offsetstable** -- The list of output diagnostic phase offset tables for the calibration targets. Defaults to the standard pipeline naming convention.

Example: `offsetstable=['M82.offsets.gacal', 'M82B.offsets.gacal']`

- **targetphasetable** -- The list of output phase calibration tables for the science targets. Defaults to the standard pipeline naming convention.

Example: `targetphasetable=['M82.gpcal', 'M82B.gpcal']`

- **amptable** -- The list of output calibration amplitude tables for the calibration and science targets. Defaults to the standard pipeline naming convention.

Example: `amptable=['M82.gacal', 'M82B.gacal']`

- **field** -- The list of field names or field ids for which gain solutions are to be computed. Defaults to all fields with the standard intent.

Example: `field='3C279', field='3C279', M82'`

- **spw** -- The list of spectral windows and channels for which gain solutions are computed. Defaults to all science spectral windows.

Example: `spw='11', spw='11,13'`

- **antenna** -- The selection of antennas for which gains are computed. Defaults to all.

- **calsolint** -- Time solution interval in CASA syntax for calibrator source solutions.

Example: `calsolint='inf', calsolint='int', calsolint='100sec'`

- **targetsolint** -- Time solution interval in CASA syntax for target source solutions.

Example: `targetsolint='inf', targetsolint='int', targetsolint='100sec'`

- **refant** -- Reference antenna name(s) in priority order. Defaults to most recent values set in the pipeline context. If no reference antenna is defined in the pipeline context use the CASA defaults.

Example: `refant='DV01', refant='DV05,DV07'`

- **refantmode** -- Controls how the refant is applied. Currently available choices are 'flex', 'strict', and the default value of ". Setting to " allows the pipeline to select the appropriate mode based on the state of the reference antenna list.

Examples: `refantmode='strict', refantmode=''`

- **solnorm** -- Normalise the gain solutions.

- **minblperant** -- Minimum number of baselines required per antenna for each solve. Antennas with fewer baselines are excluded from solutions.

Example: `minblperant=2`

- **calminsnr** -- Solutions below this SNR are rejected for calibrator solutions.

- **targetminsnr** -- Solutions below this SNR are rejected for science target solutions.

- **smodel** -- Point source Stokes parameters for source model (experimental) Defaults to using standard MODEL\_DATA column data.

Example: `smodel=[1,0,0,0]` - (I=1, unpolarized)

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`

Default: `None` (equivalent to `False`)

#### Returns

The results object for the pipeline task is returned.

### Examples

1. Compute standard per scan gain solutions that will be used to calibrate the target:

```
>>> hifa_timegaincal()
```

## 3.24 pipeline.hifa.cli.hifa\_tsysflag

```
hifa_tsysflag(vis=None, caltable=None, flag_nmedian=None, fnm_limit=None, fnm_byfield=None, flag_derivative=None, fd_max_limit=None, flag_edgechans=None, fe_edge_limit=None, flag_fieldshape=None, ff_refintent=None, ff_max_limit=None, flag_birdies=None, fb_sharps_limit=None, flag_toomany=None, tmfI_limit=None, tmefl_limit=None, metric_order=None, normalize_tsys=None, filetemplate=None, parallel=None) → ResultsList[Results]
```

Flag deviant system temperatures for ALMA interferometry measurements.

This task flags all deviant system temperature measurements in the system temperature calibration table by running a sequence of flagging tests, each designed to look for a different type of possible error.

If a file with manual Tsys flags is provided with the `filetemplate` parameter, then these flags are applied prior to the evaluation of the flagging heuristics listed below.

The tests are:

1. Flag Tsys spectra with high median values
2. Flag Tsys spectra with high median derivatives. This is meant to spot spectra that are 'ringing'.
3. Flag the edge channels of the Tsys spectra in each SpW.
4. Flag Tsys spectra whose shape is different from that associated with the BANDPASS intent.
5. Flag 'birdies'.
6. Flag the Tsys spectra of all antennas in a timestamp and spw if proportion of antennas already flagged in this timestamp and spw exceeds a threshold, and flag Tsys spectra for all antennas and all timestamps in a spw, if proportion of antennas that are already entirely flagged in all timestamps exceeds a threshold.

#### Parameters

- **vis** -- List of input MeasurementSets (Not used).
- **caltable** -- List of input Tsys calibration tables. Default: [] - Use the table currently stored in the pipeline context.  
Example: `caltable=['X132.ms.tsys.s2.tbl']`
- **flag\_nmedian** -- True to flag Tsys spectra with high median value.

- **fnm\_limit** -- Flag spectra with median value higher than fnm\_limit \* median of this measure over all spectra.
- **fnm\_byfield** -- Evaluate the nmedian metric separately for each field.
- **flag\_derivative** -- True to flag Tsys spectra with high median derivative.
- **fd\_max\_limit** -- Flag spectra with median derivative higher than fd\_max\_limit \* median of this measure over all spectra.
- **flag\_edechans** -- True to flag edges of Tsys spectra.
- **fe\_edge\_limit** -- Flag channels whose channel to channel difference > fe\_edge\_limit \* median across spectrum.
- **flag\_fieldshape** -- True to flag Tsys spectra with a radically different shape to those of the ff\_refintent.
- **ff\_refintent** -- Data intent that provides the reference shape for 'flag\_fieldshape'.
- **ff\_max\_limit** -- Flag Tsys spectra with 'fieldshape' metric values > ff\_max\_limit.
- **flag\_birdies** -- True to flag channels covering sharp spectral features.
- **fb\_sharps\_limit** -- Flag channels bracketing a channel to channel difference > fb\_sharps\_limit.
- **flag\_toomany** -- True to flag Tsys spectra for which a proportion of antennas for given timestamp and/or proportion of antennas that are entirely flagged in all timestamps exceeds their respective thresholds.
- **tmf1\_limit** -- Flag Tsys spectra for all antennas in a timestamp and spw if proportion of antennas already flagged in this timestamp and spw exceeds tmf1\_limit.
- **tmeff1\_limit** -- Flag Tsys spectra for all antennas and all timestamps in a spw, if proportion of antennas that are already entirely flagged in all timestamps exceeds tmeff1\_limit.
- **metric\_order** -- Order in which to evaluate the flagging metrics that are enabled. Disabled metrics are skipped.
- **normalize\_tsys** -- True to create a normalized Tsys table that is used to evaluate the Tsys flagging metrics. All newly found flags are also applied to the original Tsys caltable that continues to be used for subsequent calibration.
- **filetemplate** -- The name of a text file that contains the manual Tsys flagging template. If the template flags file is undefined, a name of the form 'msname.flagstemplate.txt' is assumed.
- **parallel** -- The description is missing.

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Flag Tsys measurements using currently recommended tests:

```
>>> hifa_tsysflag()
```

2. Flag Tsys measurements using all recommended tests apart from that using the 'fieldshape' metric:

```
>>> hifa_tsysflag(flag_fieldshape=False)
```

## 3.25 pipeline.hifa.cli.hifa\_tsysflagcontamination

```
hifa_tsysflagcontamination(vis=None, caltable=None, filetemplate=None, logpath=None, remove_n_extreme=None, relative_detection_factor=None, diagnostic_plots=None, continue_on_failure=None, parallel=None) → ResultsList[Results]
```

Flag line contamination in ALMA interferometric Tsys caltables.

This task flags all line contamination detected through an analysis of the Tsys and bandpass caltables.

The general idea for the detection algorithm is to discern features which appear in the Tsys calibration tables of the scans taken in the vicinity of the source field in comparison with the Tsys calibration tables of the scans taken toward the bandpass. The bandpass scan should be clean of astrophysical line features.

### Parameters

- **vis** -- List of input MeasurementSets (Not used).
- **caltable** -- List of input Tsys calibration tables. Default: [] - Use the table currently stored in the pipeline context.  
Example: caltable=['X132.ms.tsys.s2.tbl']
- **filetemplate** -- output file to which regions to flag will be written
- **logpath** -- output file to which heuristic log statements will be written
- **remove\_n\_extreme** -- expert parameter for contamination heuristic  
Default: 2
- **relative\_detection\_factor** -- expert parameter for contamination detection heuristic  
Default: 0.005
- **diagnostic\_plots** -- create diagnostic plots for the line contamination heuristic  
Default: True
- **continue\_on\_failure** -- controls whether pipeline execution continues if a failure occurs in the underlying contamination detection heuristic.  
Default: True
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: 'automatic', 'true', 'false', True, False  
Default: **None** (equivalent to **False**)

### Returns

The results object for the pipeline task is returned.

### Examples

1. Flag Tsys line contamination using currently recommended parameters:

```
>>> hifa_tsysflagcontamination()
```

2. Halt pipeline execution if a failure occurs in the underlying heuristic:

```
>>> hifa_tsysflagcontamination(continue_on_failure=False)
```

## 3.26 pipeline.hifa.cli.hifa\_unlock\_refant

**hifa\_unlock\_refant**(*vis=None*) → ResultsList[Results]

Unlock reference antenna list.

hifa\_unlock\_refant marks the reference antenna list as "unlocked" for specified MeasurementSets, allowing the list to be modified by subsequent tasks.

After executing hifa\_unlock\_refant, all subsequent gaincal calls will by default be executed with refantmode='flex'.

The refant list can be locked with the hifa\_lock\_refant task.

### Parameters

**vis** -- List of input MeasurementSets. Defaults to the list of MeasurementSets specified in the pipeline context.

Example: `vis=['ngc5921.ms']`

### Returns

The results object for the pipeline task is returned.

### Examples

1. Unlock the refant list for all MSes in pipeline context:

```
>>> hifa_unlock_refant()
```

## 3.27 pipeline.hifa.cli.hifa\_wvrgcal

**hifa\_wvrgcal**(*vis=None, caltable=None, offsetstable=None, hm\_toffset=None, toffset=None, segsource=None, sourceflag=None, hm\_tie=None, tie=None, nsol=None, disperse=None, wvrflag=None, hm\_smooth=None, smooth=None, scale=None, maxdistrm=None, minnumants=None, mingoodfrac=None, refant=None, qa\_intent=None, qa\_bandpass\_intent=None, qa\_spw=None, accept\_threshold=None*) → ResultsList[Results]

Generate a gain table based on Water Vapor Radiometer (WVR) data.

Generate a gain table based on the Water Vapor Radiometer data in each vis file. By applying the wvr calibration to the data specified by `qa_intent` and `qa_spw`, calculate a QA score to indicate its effect on interferometric data; a score > 1 implies that the phase noise is improved, a score < 1 implies that it is made worse. If the score is less than `accept_threshold` then the wvr gain table is not accepted into the context for subsequent use.

### Parameters

- **vis** -- List of input visibility files. Default: `None`, in which case the vis files to be used will be read from the context.

Example: `vis=['ngc5921.ms']`

- **caltable** -- List of output gain calibration tables. Default: `None`, in which case the names of the caltables will be generated automatically.

Example: `caltable='ngc5921.wvr'`

- **offsetstable** -- List of input temperature offsets table files to subtract from WVR measurements before calculating phase corrections. Default: `None`, in which case no offsets are applied.

Example: `offsetstable=['ngc5921.cloud_offsets']`

- **hm\_toffset** -- If 'manual', set the `toffset` parameter to the user-specified value. If 'automatic', set the `toffset` parameter according to the date of the MeasurementSet; `toffset=-1` if before 2013-01-21T00:00:00 `toffset=0` otherwise.

- **toffset** -- Time offset (sec) between interferometric and WVR data.
- **segsource** -- If True calculate new atmospheric phase correction coefficients for each source, subject to the constraints of the **tie** parameter. 'segsource' is forced to be True if the **tie** parameter is set to a non-empty value by the user or by the automatic heuristic.
- **sourceflag** -- Flag the WVR data for these source(s) as bad and do not produce corrections for it. Requires **segsource** = True.

Example: `['3C273']`

- **hm\_tie** -- If 'manual', set the **tie** parameter to the user-specified value. If 'automatic', set the **tie** parameter to include with the target all calibrators that are within 15 degrees of it: if no calibrators are that close then **tie** is left empty.
- **tie** -- Use the same atmospheric phase correction coefficients when calculating the WVR correction for all sources in the **tie**. If **tie** is not empty then **segsource** is forced to be True. Ignored unless **hm\_tie** = 'manual'.

Example: `tie=['3C273,NGC253', 'IC433,3C279']`

- **nsol** -- Number of solutions for phase correction coefficients during this observation, evenly distributed in time throughout the observation. It is used only if **segsource** = False because if **segsource** = True then the coefficients are recomputed whenever the telescope moves to a new source (within the limits imposed by **tie**).
- **disperse** -- Apply correction for dispersion. (Deprecated; will be removed)
- **wvrflag** -- Flag the WVR data for the listed antennas as bad and replace their data with values interpolated from the 3 nearest antennas with unflagged data.

Example: `['DV03', 'DA05', 'PM02']`

- **hm\_smooth** -- If 'manual' set the **smooth** parameter to the user-specified value. If 'automatic', run the **wvrgcal** task with the range of **smooth** parameters required to match the integration time of the wvr data to that of the interferometric data in each spectral window.
- **smooth** -- Smooth WVR data on this timescale before calculating the correction. Ignored unless **hm\_smooth**='manual'.
- **scale** -- Scale the entire phase correction by this factor.

- **maxdistm** -- Maximum distance in meters of an antenna used for interpolation from a flagged antenna. Default: -1 (automatically set to 100m if >50% of antennas are 7m antennas without WVR and otherwise set to 500m).

Example: `maxdistm=550`

- **minnumants** -- Minimum number of nearby antennas (up to 3) used for interpolation from a flagged antenna.

Example: `minnumants=3`

- **mingoodfrac** -- Minimum fraction of good data per antenna.

- **refant** -- Ranked comma delimited list of reference antennas.

Example: `refant='DV01,DV02'`

- **qa\_intent** -- The list of data intents on which the wvr correction is to be tried as a means of estimating its effectiveness. A QA 'view' will be calculated for each specified intent, in each spectral window in each vis file. Each QA 'view' will consist of a pair of 2-d images with dimensions ['ANTENNA', 'TIME'], one showing the data phase-noise before the wvr application, the second showing the phase noise after (both 'before' and 'after' images have a bandpass calibration applied as well). An overall QA score is calculated for each vis file, by dividing the 'before' images by the 'after' and taking the median of the result. An overall score of 1 would correspond to no change in the phase noise, a score > 1 implies an improvement. If the overall score for a vis file is less than the value in 'accept\_threshold' then the wvr calibration file is not made available for merging into the context for use in the subsequent reduction. If you do not want any QA calculations then set **qa\_intent**=".

Example: `qa_intent='PHASE'`

- **`qa_bandpass_intent`** -- The data intent to use for the bandpass calibration in the qa calculation. The default is blank to allow the underlying bandpass task to select a sensible intent if the dataset lacks BANDPASS data.
- **`qa_spw`** -- The SpW(s) to use for the qa calculation, in the order that they should be tried. Input as a comma-separated list. The default is blank, in which case the task will try SpWs in order of decreasing median sky opacity.
- **`accept_threshold`** -- The phase-rms improvement ratio (rms without wvr / rms with wvr) above which the wrvg file will be accepted into the context for subsequent application.

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Compute the WVR calibration for all the MeasurementSets:

```
>>> hifa_wvrgcal(hm_tie='automatic')
```

## 3.28 pipeline.hifa.cli.hifa\_wvrgcalflag

**`hifa_wvrgcalflag`**(vis=None, caltable=None, offsetstable=None, hm\_toffset=None, toffset=None, segsource=None, sourceflag=None, hm\_tie=None, tie=None, nsol=None, disperse=None, wvrflag=None, hm\_smooth=None, smooth=None, scale=None, maxdistrm=None, minnumants=None, mingoodfrac=None, refant=None, flag\_intent=None, qa\_intent=None, qa\_bandpass\_intent=None, accept\_threshold=None, flag\_hi=None, fhi\_limit=None, fhi\_minsample=None, ants\_with\_wvr\_thresh=None, parallel=None) → ResultsList[Results]

Flag bad WVR calibration in gain table and interpolate over antennas with bad radiometers.

This task will first identify for each vis whether it includes at least 3 antennas with Water Vapor Radiometer (WVR) data, and that the fraction of WVR antennas / all antennas exceeds the minimum threshold (ants\_with\_wvr\_thresh).

If there are not enough WVR antennas by number and/or fraction, then no WVR caltable is created and no WVR calibration will be applied to the corresponding vis. If there are enough WVR antennas, then the task proceeds as follows for each valid vis:

First, generate a gain table based on the Water Vapor Radiometer data for each vis.

Second, apply the WVR calibration to the data specified by 'flag\_intent', calculate flagging 'views' showing the ratio 'phase-rms with WVR / phase-rms without WVR' for each scan. A ratio < 1 implies that the phase noise is improved, a ratio > 1 implies that it is made worse.

Third, search the flagging views for antennas with anomalous high values. If any are found then recalculate the WVR calibration with the 'wvrflag' parameter set to ignore their data and interpolate results from other antennas according to 'maxdistrm' and 'minnumants'.

Fourth, after flagging, if the remaining unflagged antennas with WVR number fewer than 3, or represent a smaller fraction of antennas than the minimum threshold (ants\_with\_wvr\_thresh), then the WVR calibration file is rejected and will not be merged into the context, i.e. not be used in subsequent calibration.

Fifth, if the overall QA score for the final WVR correction of a vis file is greater than the value in 'accept\_threshold' then make available the wvr calibration file for merging into the context and use in the subsequent reduction.

#### Parameters

- **`vis`** -- List of input visibility files. Default: none, in which case the vis files to be used will be read from the context.

Example: `vis=['ngc5921.ms']`

- **caltable** -- List of output gain calibration tables. Default: none, in which case the names of the caltables will be generated automatically.

Example: `caltable='ngc5921.wvr'`

- **offsetstable** -- List of input temperature offsets table files to subtract from WVR measurements before calculating phase corrections. Default: none, in which case no offsets are applied.

Example: `offsetstable=['ngc5921.cloud_offsets']`

- **hm\_toffset** -- If 'manual', set the 'toffset' parameter to the user-specified value. If 'automatic', set the 'toffset' parameter according to the date of the MeasurementSet; toffset=-1 if before 2013-01-21T00:00:00 toffset=0 otherwise.

- **toffset** -- Time offset (sec) between interferometric and WVR data.

- **segsource** -- If True calculate new atmospheric phase correction coefficients for each source, subject to the constraints of the **tie** parameter. **segsource** is forced to be True if the **tie** parameter is set to a non-empty value by the user or by the automatic heuristic.

- **sourceflag** -- Flag the WVR data for these source(s) as bad and do not produce corrections for it. Requires `segsource=True`.

Example: `sourceflag=['3C273']`

- **hm\_tie** -- If 'manual', set the **tie** parameter to the user-specified value. If 'automatic', set the **tie** parameter to include with the target all calibrators that are within 15 degrees of it: if no calibrators are that close then **tie** is left empty.

- **tie** -- Use the same atmospheric phase correction coefficients when calculating the WVR correction for all sources in the **tie**. If **tie** is not empty then **segsource** is forced to be True. Ignored unless `hm_tie = 'manual'`.

Example: `tie=['3C273', 'NGC253', 'IC433', '3C279']`

- **nsol** -- Number of solutions for phase correction coefficients during this observation, evenly distributed in time throughout the observation. It is used only if `segsource=False` because if `segsource=True` then the coefficients are recomputed whenever the telescope moves to a new source (within the limits imposed by '**tie**').

- **disperse** -- Apply correction for dispersion. (Deprecated; will be removed)

- **wvrflag** -- Flag the WVR data for these antenna(s) as bad and replace its data with interpolated values.

Example: `wvrflag=['DV03', 'DA05', 'PM02']`

- **hm\_smooth** -- If 'manual' set the 'smooth' parameter to the user-specified value. If 'automatic', run the `wvrgcal` task with the range of 'smooth' parameters required to match the integration time of the WVR data to that of the interferometric data in each spectral window.

- **smooth** -- Smooth WVR data on this timescale before calculating the correction. Ignored unless `hm_smooth='manual'`.

- **scale** -- Scale the entire phase correction by this factor.

- **maxdistm** -- Distance in meters of an antenna used for interpolation from a flagged antenna. Default: -1 (automatically set to 100m if >50% of antennas are 7m antennas without WVR and otherwise set to 500m).

Example: `maxdistm=550`

- **minnumants** -- Minimum number of nearby antennas (up to 3) used for interpolation from a flagged antenna.

Example: `minnumants=3`

- **mingoodfrac** -- Minimum fraction of good data per antenna.

Example: `mingoodfrac=0.7`

- **refant** -- Ranked comma delimited list of reference antennas.

Example: `refant='DV02,DV06'`

- **flag\_intent** -- The data intent(s) on whose WVR correction results the search for bad WVR antennas is to be based. A 'flagging view' will be calculated for each specified intent, in each spectral window in each vis file. Each 'flagging view' will consist of a 2-d image with dimensions ['ANTENNA', 'TIME'], showing the phase noise after the WVR correction has been applied. If flag\_intent is left blank, the default, the flagging views will be derived from data with the default bandpass calibration intent i.e. the first in the list BANDPASS, PHASE, AMPLITUDE for which the MeasurementSet has data.
- **qa\_intent** -- The list of data intents on which the WVR correction is to be tried as a means of estimating its effectiveness. A QA 'view' will be calculated for each specified intent, in each spectral window in each vis file. Each QA 'view' will consist of a pair of 2-d images with dimensions ['ANTENNA', 'TIME'], one showing the data phase-noise before the WVR application, the second showing the phase noise after (both 'before' and 'after' images have a bandpass calibration applied as well). An overall QA score is calculated for each vis file, by dividing the 'before' images by the 'after' and taking the median of the result. An overall score of 1 would correspond to no change in the phase noise, a score > 1 implies an improvement. If the overall score for a vis file is less than the value in 'accept\_threshold' then the WVR calibration file is not made available for merging into the context for use in the subsequent reduction.
- **qa\_bandpass\_intent** -- The data intent to use for the bandpass calibration in the qa calculation. The default is blank to allow the underlying bandpass task to select a sensible intent if the dataset lacks BANDPASS data.
- **accept\_threshold** -- The phase-rms improvement ratio (rms without WVR / rms with WVR) above which the wrvg file will be accepted into the context for subsequent application.
- **flag\_hi** -- True to flag high figure of merit outliers.
- **fhi\_limit** -- Flag figure of merit values higher than limit \* MAD.
- **fhi\_minsample** -- Minimum number of samples for valid MAD estimate.
- **ants\_with\_wvr\_thresh** -- This threshold sets the minimum fraction of antennas that should have WVR data for WVR calibration and flagging to proceed; the same threshold is used to determine, after flagging, whether there remain enough unflagged antennas with WVR data for the WVR calibration to be applied. Example: `ants_with_wvr_thresh=0.5`
- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.

Options: `'automatic', 'true', 'false', True, False`

Default: `None` (equivalent to `False`)

## Returns

The results object for the pipeline task is returned.

## Examples

1. Compute the WVR calibration for all the MeasurementSets:

```
>>> hifa_wvrgcalflag(hm_tie='automatic')
```

---

**CHAPTER  
FOUR**

---

**PIPELINE.HIFV.CLI**

Interferometry VLA Tasks

**Functions**

<i>hifv_analyzestokescubes</i>	Characterize stokes IQUV flux densities as a function of frequency for VLASS Coarse Cube (CC) images.
<i>hifv_applycals</i>	Apply calibration tables to input MeasurementSets.
<i>hifv_checkflag</i>	Run RFI flagging using flagdata in various modes.
<i>hifv_circfeedpolcal</i>	Perform polarization calibration for VLA circular feeds.
<i>hifv_exportdata</i>	Prepare and export interferometry and imaging data.
<i>hifv_exportvllassdata</i>	Export Image data from QL, SE, and Coarse Cube modes of VLASS Survey.
<i>hifv_finalcals</i>	Compute final gain calibration tables.
<i>hifv_fixpointing</i>	Base fixpointing task.
<i>hifv_flagcal</i>	Flagcal task.
<i>hifv_flagdata</i>	Do basic deterministic flagging.
<i>hifv_flagtargetsdata</i>	Apply a flagtemplate to target data prior to running imaging pipeline tasks.
<i>hifv_fluxboot</i>	Determine flux density bootstrapping for gain calibrators relative to flux calibrator.
<i>hifv_hanning</i>	
<i>hifv_importdata</i>	Imports data into the VLA pipeline.
<i>hifv_mstransform</i>	Create new MeasurementSets for science target imaging.
<i>hifv_pbcor</i>	Apply primary beam correction to VLA and VLASS images.
<i>hifv_plotssummary</i>	Create pipeline summary plots.
<i>hifv_priorcals</i>	Runs gaincurves, opacities, requantizer gains, antenna position corrections, tec_maps, switched power.
<i>hifv_restoredata</i>	Restore flagged and calibration interferometry data from a pipeline run.
<i>hifv_restorepims</i>	Restore VLASS SE per-image MeasurementSet data, resetting flagging, weights, and applying self-calibration.
<i>hifv_selfcal</i>	Perform phase-only self-calibration, per scan row, on VLASS SE images.
<i>hifv_semiFinalBPdcals</i>	Runs a second delay and bandpass calibration and applies to calibrators to setup for RFI flagging.
<i>hifv_solint</i>	Determines different solution intervals.
<i>hifv_statwt</i>	Compute statistical weights and write them to measurement set.

continues on next page

Table 1 – continued from previous page

<code>hifv_syspower</code>	Determine amount of gain compression affecting VLA data below Ku-band.
<code>hifv_testBPdcals</code>	Runs initial delay and bandpass calibration to setup for RFI flagging.
<code>hifv_vlasetjy</code>	Sets flux density scale and fills calibrator model to MeasurementSets.
<code>hifv_vlassmasking</code>	Create clean masks for VLASS Single Epoch (SE) images.

## 4.1 pipeline.hifv.cli.hifv\_analyzestokescubes

`hifv_analyzestokescubes(vis=None)` → Results

Characterize stokes IQUV flux densities as a function of frequency for VLASS Coarse Cube (CC) images.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.

### Returns

The results object for the pipeline task is returned.

### Examples

1. Basic analyzestokescubes task

```
>>> hifv_analyzestokescubes()
```

## 4.2 pipeline.hifv.cli.hifv\_applycals

`hifv_applycals(vis=None, field=None, intent=None, spw=None, antenna=None, applymode=None, flagbackup=None, flagsum=None, flagdetailedsum=None, gainmap=None)` → ResultsList[Results]

Apply calibration tables to input MeasurementSets.

hifv\_applycals applies the precomputed calibration tables stored in the pipeline context to the set of visibility files using predetermined field and spectral window maps and default values for the interpolation schemes.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **field** -- A string containing the list of field names or field ids to which the calibration will be applied. Defaults to all fields in the pipeline context.

Example: '3C279', '3C279', M82'

- **intent** -- A string containing the list of intents against which the selected fields will be matched. Defaults to all supported intents in the pipeline context.

Example: '\*TARGET\*'

- **spw** -- The list of spectral windows and channels to which the calibration will be applied. Defaults to all science windows in the pipeline.

Example: '17', '11, 15'

- **antenna** -- The selection of antennas to which the calibration will be applied. Defaults to all antennas. Not currently supported.
- **applymode** -- Calibration apply mode.
  - 'calflag': calibrate data and apply flags from solutions
  - 'calflagstrict': same as above except flag spws for which calibration is unavailable in one or more tables (instead of allowing them to pass uncalibrated and unflagged)
  - 'trial': report on flags from solutions, dataset entirely unchanged
  - 'flagonly': apply flags from solutions only, data not calibrated
  - 'flagonlystrict': same as above except flag spws for which calibration is unavailable in one or more tables
  - 'calonly': calibrate data only, flags from solutions NOT applied
- **flagbackup** -- Backup the flags before the apply.
- **flagsum** -- Compute before and after flagging summary statistics.
- **flagdetailedsum** -- Compute detailed flagging statistics.
- **gainmap** -- Mode to map gainfields to scans.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Run the final applycals stage of the VLA CASA pipeline:

```
>>> hifv_applycals()
```

## 4.3 pipeline.hifv.cli.hifv\_checkflag

**hifv\_checkflag**(vis=None, checkflagmode=None, growflags=None, overwrite\_modelcol=None) → ResultsList[Results]

Run RFI flagging using flagdata in various modes.

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **checkflagmode** --
  - Standard VLA modes with improved RFI flagging heuristics: 'bpv-vla', 'allcals-vla', 'target-vla'
  - blank string default use of rflag on bandpass and delay calibrators
  - use string 'semi' after hifv\_semiFinalBPdcals() for executing rflag on calibrators
  - use string 'bpv', for the bandpass and delay calibrators: execute rflag on all calibrated cross-hand corrected data; extend flags to all correlations execute rflag on all calibrated parallel-hand residual data; extend flags to all correlations execute tfcrop on all calibrated cross-hand corrected data, per visibility; extend flags to all correlations execute tfcrop on all calibrated parallel-hand corrected data, per visibility; extend flags to all correlations
  - use string 'allcals', for all the other calibrators, with delays and BPcal applied: similar procedure as 'bpv' mode, but uses corrected data throughout

- use string 'target', for the target data: similar procedure as 'allcals' mode, but with a higher SNR cutoff for rflag to avoid flagging data due to source structure, and with an additional series of tfcrop executions to make up for the higher SNR cutoff in rflag
- VLASS specific modes include 'bpd-vlass', 'allcals-vlass', and 'target-vlass' which calculate thresholds to use per spw/field/scan (action='calculate'), then, per baseband/field/scan, replace all spw thresholds above the median with the median, before re-running rflag with the new thresholds. This has the effect of lowering the thresholds for spws with RFI to be closer to the RFI-free thresholds, and catches more of the RFI.
- Mode 'vlass-imaging' is similar to 'target-vlass', except that it executes on the split off target data, intent='\*TARGET', datacolumn='data' and uses a timedevscale of 4.0.
- **growflags** -- Grow flags in time at the end of the following checkflagmodes:
  - default=True, for 'bpd-vla', 'allcals-vla', 'bpd', and 'allcals.'
  - default=False, for " and 'semi'
- **overwrite\_modelcol** -- Always write the model column, even if it already exists.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Run RFLAG with associated heuristics in the VLA CASA pipeline:

```
>>> hifv_checkflag()
```

## 4.4 pipeline.hifv.cli.hifv\_circfeedpolcal

**hifv\_circfeedpolcal**(vis=None, Dterm\_solt=None, refantignore=None, leakage\_poltype=None, mbdkcross=None, clipminmax=None, refant=None, run\_setjy=None) → ResultsList[Results]

Perform polarization calibration for VLA circular feeds.

Only validated for VLA sky survey data in S-band continuum mode with 3C138 or 3C286 as polarization angle. Requires that all polarization intents are properly set during observation.

**Parameters**

- **vis** -- List of input visibility data.
- **Dterm\_solt** -- D-terms spectral averaging.  
Example: refantignore='ea02,ea03'.
- **refantignore** -- String list of antennas to ignore.
- **leakage\_poltype** -- poltype to use in first polcal execution - blank string means use default heuristics.
- **mbdkcross** -- Run gaincal KCROSS grouped by baseband.
- **clipminmax** -- Acceptable range for leakage amplitudes, values outside will be flagged.
- **refant** -- A csv string of reference antenna(s). When used, disables **refantignore**. Example: refant = 'ea01, ea02'
- **run\_setjy** -- Run setjy for amplitude/flux calibrator, default set to True.

**Returns**

The results object for the pipeline task is returned.

## Examples

1. Basic circfeedpolcal task:

```
>>> hifv_circfeedpolcal()
```

## 4.5 pipeline.hifv.cli.hifv\_exportdata

**hifv\_exportdata**(vis=None, session=None, imaging\_products\_only=None, exportmses=None, tarms=None, exportcalprods=None, pprfile=None, calintents=None, calimages=None, targetimages=None, products\_dir=None, gainmap=None) → Results

Prepare and export interferometry and imaging data.

The hifv\_exportdata task for the VLA CASA pipeline exports the data defined in the pipeline context and exports it to the data products directory, converting and or packing it as necessary.

The current version of the task exports the following products

- an XML file containing the pipeline processing request
- a tar file per ASDM / MS containing the final flags version OR the MS if tarms is False
- a text file per ASDM / MS containing the final calibration apply list
- a FITS image for each selected calibrator source image
- a FITS image for each selected science target source image
- a tar file per session containing the caltables for that session
- a tar file containing the file web log
- a text file containing the final list of CASA commands

### Parameters

- **vis** -- List of visibility data files for which flagging and calibration information will be exported. Defaults to the list maintained in the pipeline context. example: `vis=['X227.ms', 'X228.ms']`
- **session** -- List of sessions one per visibility file. Currently defaults to a single virtual session containing all the visibility files in vis. In the future, this will default to the set of observing sessions defined in the context. example: `session=['session1', 'session2']`
- **imaging\_products\_only** -- Export science target imaging products only
- **exportmses** -- Export the final MeasurementSets instead of the final flags, calibration tables, and calibration instructions.
- **tarms** -- Tar final MeasurementSets
- **exportcalprods** -- Export flags and caltables in addition to MeasurementSets. this parameter is only valid when exportmses = True.
- **pprfile** -- Name of the pipeline processing request to be exported. Defaults to a file matching the template 'PPR\_\*.xml'. example: `pprfile=['PPR_GRB021004.xml']`
- **calintents** -- List of calibrator image types to be exported. Defaults to all standard calibrator intents, 'BAND-PASS', 'PHASE', 'FLUX'. example: `'PHASE'`
- **calimages** -- List of calibrator images to be exported. Defaults to all calibrator images recorded in the pipeline context. example: `calimages=['3C454.3.bandpass', '3C279.phase']`

- **targetimages** -- List of science target images to be exported. Defaults to all science target images recorded in the pipeline context. example: `targetimages=['NGC3256.band3', 'NGC3256.band6']`
- **products\_dir** -- Name of the data products subdirectory. Defaults to './' example: `'./products'`
- **gainmap** -- The value of `gainmap` parameter in `hifv_restoredata` task put in `casa_piperestorescript.py`

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Export the pipeline results for a single session to the data products directory:

```
>>> !mkdir ../products
>>> hifv_exportdata (products_dir='..../products')
```

2. Export the pipeline results to the data products directory specify that only the gain calibrator images be saved:

```
>>> !mkdir ../products
>>> hifv_exportdata (products_dir='..../products', calintents='*PHASE*')
```

## 4.6 pipeline.hifv.cli.hifv\_exportvllassdata

**hifv\_exportvllassdata**(*vis=None*) → ResultsList[Results]

Export Image data from QL, SE, and Coarse Cube modes of VLASS Survey.

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the `hifv_importdata` task.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Basic exportvllassdata task:

```
>>> hifv_exportvllassdata()
```

## 4.7 pipeline.hifv.cli.hifv\_finalcals

**hifv\_finalcals**(*vis=None, weakbp=None, refantignore=None, refant=None*) → ResultsList[Results]

Compute final gain calibration tables.

**Parameters**

- **vis** (*str, optional*) -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the `hifv_importdata` task.
- **weakbp** (*Boolean*) -- Activate weak bandpass heuristics. weak bandpass heuristics on/off - currently not used - see PIPE-104.
- **refantignore** (*str*) -- String list of antennas to ignore. csv string of reference antennas to ignore - 'ea24,ea15,ea08'.

- **refant** (*List*) -- A csv string of reference antenna(s). When used, disables **refantignore**.

Example: refant = 'ea01, ea02'

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Create the final calibration tables to be applied to the data in the VLA CASA pipeline:

```
>>> hifv_finalcals()
```

## 4.8 pipeline.hifv.cli.hifv\_fixpointing

**hifv\_fixpointing**(*vis=None*) → ResultsList[Results]

Base fixpointing task.

#### Parameters

**vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the **hifv\_importdata** task.

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Basic fixpointing task:

```
>>> hifv_fixpointing()
```

## 4.9 pipeline.hifv.cli.hifv\_flagcal

**hifv\_flagcal**(*vis=None, caltable=None, clipminmax=None*) → ResultsList[Results]

Flagcal task.

#### Parameters

- **vis** -- List of input visibility data.
- **caltable** -- String name of the caltable.
- **clipminmax** -- Range to use for clipping.

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Flag existing caltable:

```
>>> hifv_flagcal()
```

## 4.10 pipeline.hifv.cli.hifv\_flagdata

**hifv\_flagdata**(*vis=None*, *autocorr=None*, *shadow=None*, *scan=None*, *scannumber=None*, *quack=None*, *clip=None*, *baseband=None*, *intents=None*, *edgespw=None*, *fracspw=None*, *online=None*, *fileonline=None*, *template=None*, *filetemplate=None*, *hm\_tbuff=None*, *tbuff=None*, *flagbackup=None*) → ResultsList[Results]

Do basic deterministic flagging.

The hifv\_flagdata task performs basic flagging operations on a list of MeasurementSets including:

- autocorrelation data flagging
- shadowed antenna data flagging
- scan based flagging
- edge channel flagging
- baseband edge flagging
- applying online flags
- applying a flagging template
- quack, shadow, and basebands
- Antenna not-on-source (ANOS)

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **autocorr** -- Flag autocorrelation data
- **shadow** -- Flag shadowed antennas
- **scan** -- Flag specified scans
- **scannumber** -- A string containing a comma delimited list of scans to be flagged.  
Example: '**3,5,6**'
- **quack** -- Quack scans
- **clip** -- Clip mode
- **baseband** -- Flag 20MHz of each edge of basebands
- **intents** -- A string containing a comma delimited list of intents against which the scans to be flagged are matched.  
Example: '**\*BANDPASS\***'
- **edgespw** -- Fraction of the baseline correlator TDM edge channels to be flagged.
- **fracspw** -- Fraction of baseline correlator edge channels to be flagged.
- **online** -- Apply the online flags.
- **fileonline** -- File containing the online flags. These are computed by the h\_init or hifv\_importdata data tasks. If the online flags files are undefined a name of the form 'msname.flagonline.txt' is assumed.
- **template** -- Apply a flagging template.
- **filetemplate** -- The name of a text file that contains the flagging template for RFI, birdies, telluric lines, etc. If the template flags files is undefined a name of the form 'msname.flagtemplate.txt' is assumed.

- **hm\_tbuff** -- The time buffer computation heuristic
- **tbuff** -- List of time buffers (sec) to pad timerange in flag commands.
- **flagbackup** -- Backup pre-existing flags before applying new ones.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Do basic flagging on a MeasurementSet:

```
>>> hifv_flagdata()
```

2. Do basic flagging on a MeasurementSet as well as flag pointing and atmosphere data:

```
>>> hifv_flagdata(scan=True intent='*BANDPASS*')
```

## 4.11 pipeline.hifv.cli.hifv\_flagtargetsdata

**hifv\_flagtargetsdata**(vis=None, template=None, filetemplate=None, flagbackup=None) → ResultsList[Results]

Apply a flagtemplate to target data prior to running imaging pipeline tasks.

**Parameters**

- **vis (str)** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets defined in the pipeline context.
- **template (bool)** -- Apply flagging templates.
- **filetemplate (str)** -- The name of a text file that contains the flagging template for issues with the science target data etc. If the template flags files is undefined a name of the form 'msname\_flagtargetstemplate.txt' is assumed. Flags from template will be applied to all relevant MSes.
- **flagbackup (bool)** -- Back up any pre-existing flags.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Basic flagtargetsdata task:

```
>>> hifv_flagtargetsdata()
```

## 4.12 pipeline.hifv.cli.hifv\_fluxboot

**hifv\_fluxboot**(vis=None, caltable=None, fitorder=None, refantignore=None, refant=None) → ResultsList[Results]

Determine flux density bootstrapping for gain calibrators relative to flux calibrator.

**Parameters**

- **vis (str or list)** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.

- **caltable (str)** -- fluxgaincal table from user input. If None, task uses default name. If a caltable is specified, then the fluxgains stage from the scripted pipeline is skipped and we proceed directly to the flux density bootstrapping.
- **fitorder (int)** -- Polynomial order of the spectral fitting for valid flux densities with multiple spws. The default value of -1 means that the heuristics determine the fit order based on fractional bandwidth and receiver bands present in the observation. An override value of 1,2,3 or 4 may be specified by the user. Spectral index (1) and, if applicable, curvature (2) are reported in the weblog. If no determination can be made by the heuristics, a fitorder of 1 will be used. Default is -1 (heuristics will determine).
- **refantignore (str)** -- String list of antennas to ignore  
Example: refantignore='ea02, ea03'
- **refant (str)** -- A csv string of reference antenna(s). When used, disables **refantignore**.  
Example: refant = 'ea01, ea02'

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. VLA CASA pipeline flux density bootstrapping:

```
>>> hifv_fluxboot()
```

## 4.13 pipeline.hifv.cli.hifv\_hanning

**hifv\_hanning**(vis: str | None = None, maser\_detection: bool | None = None, spws\_to\_smooth: str | None = None) → ResultsList[Results]

**Parameters**

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **maser\_detection** -- Run maser detect algorithm on spectral line windows if spws\_to\_smooth is None. Defaults to True.
- **spws\_to\_smooth** -- A CASA-style range of spw IDs indicating which ones to smooth.

Example: '1,2~4,7' indicates spws 1, 2, 3, 4, and 7 should be smoothed.

Hanning smoothing on a dataset.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Run the task to execute hanning smoothing on a VLA CASA pipeline loaded MeasurementSet:

```
>>> hifv_hanning()
```

2. Run the task with maser detection off and to only smooth spws 2 through 5.

```
>>> hifv_hanning(maser_detection=False, spws_to_smooth='2~5')
```

## 4.14 pipeline.hifv.cli.hifv\_importdata

**hifv\_importdata**(*vis=None*, *session=None*, *asis=None*, *overwrite=None*, *nocopy=None*, *createmms=None*, *ocorr\_mode=None*, *datacolumns=None*, *specline\_spws=None*, *parallel=None*) → ResultsList[Results]

Imports data into the VLA pipeline.

The **hifv\_importdata** task loads the specified visibility data into the pipeline context unpacking and / or converting it as necessary.

### Parameters

- **vis** -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSes, or tar files of MSes, If ASDM files are specified, they will be converted to MS format.

Example: `vis=['X227.ms', 'asdms.tar.gz']`

- **session** -- List of sessions to which the visibility files belong. Defaults to a single session containing all the visibility files, otherwise a session must be assigned to each vis file.

Example: `session=['Session_1', 'Sessions_2']`

- **asis** -- Creates verbatim copies of the ASDM tables in the output MS. The value given to this option must be a list of table names separated by space characters. examples:

– `asis='Receiver CalAtmosphere'`

– `asis='Receiver', asis=''`

- **overwrite** -- Overwrite existing files on import.

- **nocopy** -- When importing an MS, disable copying of the MS to the working directory.

- **createmms** -- Create a multi-MeasurementSet ('true') ready for parallel processing, or a standard MeasurementSet ('false'). The default setting ('automatic') creates an MMS if running in a cluster environment.

- **ocorr\_mode** -- Read in cross- and auto-correlation data(ca), cross- correlation data only (co), or autocorrelation data only (ao).

- **datacolumns** -- Dictionary defining the data types of existing columns. The format is:

`{'data': 'data type 1'}`

or

`{'data': 'data type 1', 'corrected': 'data type 2'}`

For ASDMs the data type can only be RAW and one can only specify it for the data column. For MSes one can define two different data types for the DATA and CORRECTED\_DATA columns and they can be any of the known data types (RAW, REGCAL\_CONTLINE\_ALL, REGCAL\_CONTLINE\_SCIENCE, SELFCAL\_CONTLINE\_SCIENCE, REGCAL\_LINE\_SCIENCE, SELFCAL\_LINE\_SCIENCE, BASELINED, ATMCORR). The intent selection strings \_ALL or \_SCIENCE can be skipped. In that case the task determines this automatically by inspecting the existing intents in the dataset. Usually, a single datacolumns dictionary is used for all datasets. If necessary, one can define a list of dictionaries, one for each EB, with different setups per EB. If no types are specified, {'data':'raw','corrected':'regcal\_contline'} or {'data':'raw'} will be assumed, depending on whether the corrected column exists or not.

- **specline\_spws** -- String indicating how the pipeline should determine whether a spw should be processed as a spectral line window or continuum. The default setting of 'auto' will use defined heuristics to determine this definition. Accepted values are 'auto', 'none' (no spws will be defined as spectral line), or a string of spw definitions in the CASA format.

Example: `specline_spws='2, 3, 4~9, 23'`

- **parallel** -- Process multiple MeasurementSets in parallel using the casampi parallelization framework.  
Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`  
Default: `None` (equivalent to `False`)

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Load an ASDM list in the `./rawdata` subdirectory into the context:

```
>>> hifv_importdata (vis=['./rawdata/uid__A002_X30a93d_X43e', './rawdata/uid_A002_x30a93d_X44e
->'])
```

2. Load an MS in the current directory into the context:

```
>>> hifv_importdata (vis=['uid__A002_X30a93d_X43e.ms'])
```

3. Load a tarred ASDM in `./rawdata` into the context:

```
>>> hifv_importdata (vis=['./rawdata/uid__A002_X30a93d_X43e.tar.gz'])
```

4. Check the `hifv_importdata` inputs, then import the data:

```
>>> myvislist = ['uid__A002_X30a93d_X43e.ms', 'uid_A002_x30a93d_X44e.ms']
>>> hifv_importdata(vis=myvislist)
```

5. Run with explicit setting of data column types:

```
>>> hifv_importdata(vis=['uid__A002_X30a93d_X43e_targets.ms'], datacolumns={'data': 'regcal_
->contline'})
>>> hifv_importdata(vis=['uid__A002_X30a93d_X43e_targets_line.ms'], datacolumns={'data': 'regcal_
->line', 'corrected': 'selfcal_line'})
```

## 4.15 pipeline.hifv.cli.hifv\_mstransform

`hifv_mstransform(vis=None, outputvis=None, outputvis_for_line=None, field=None, intent=None, spw=None, spw_line=None, chanbin=None, timebin=None, omit_contline_ms=None) → ResultsList[Results]`

Create new MeasurementSets for science target imaging.

Create new MeasurementSets for imaging from the corrected column of the input MeasurementSet via calling `mstransform` with all data selection parameters. By default, all science target data is copied to the new MS(s). The new MeasurementSet is not re-indexed to the selected data and the new MS will have the same source, field, and spw names and ids as it does in the parent MS.

The first MeasurementSet that is produced is intended for continuum imaging and will end in `targets_cont.ms`. If there are spws that have been detected or specified as spectral line spws in the input MeasurementSet, an MS for science target line imaging will also be produced, which will end in `_targets.ms`.

#### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task. ":" use all MeasurementSets in the context

Examples: `'ngc5921.ms', ['ngc5921a.ms', 'ngc5921b.ms', 'ngc5921c.ms']`

- **outputvis** -- A list of output MeasurementSets that will contain the transformed and flagged data for continuum imaging. This list must have the same length as the input list.

Default Naming: By default, an input MS named `<msrootname>.ms` will produce an output named `<msrootname>_targets_cont.ms`.

### Examples

- `outputvis='ngc5921_targets_cont.ms'`
- `outputvis=['ngc5921a_targets_cont.ms', 'ngc5921b_targets_cont.ms', 'ngc5921c_targets_cont.ms']`

- **outputvis\_for\_line** -- A list of output MeasurementSets for line detection and imaging, created without RFI flagging. This list must have the same length as the input list.

Default Naming: By default, an input MS named `<msrootname>.ms` will produce an output named `<msrootname>_targets.ms`.

### Examples

- `outputvis_for_line='ngc5921_targets.ms'`
- `outputvis_for_line=['ngc5921a_targets.ms', 'ngc5921b_targets.ms', 'ngc5921c_targets.ms']`
- **field** -- Select fields name(s) or id(s) to transform. Only fields with data matching the intent will be selected.  
Examples: `'3C279', 'Centaurus*', '3C279, J1427-421'`
- **intent** -- Select intents for which associated fields will be imaged. By default only TARGET data is selected.  
Examples: `'PHASE, BANDPASS'`
- **spw** -- Select spectral window/channels to include for continuum imaging. By default all science spws for which the specified intent is valid are selected.
- **spw\_line** -- Select spectral window/channels to include for line imaging. If specified, these will override the default, which is to use the spws identified as specline\_windows in hifv\_importdata or hifv\_restoredata.
- **chanbin** -- Width (bin) of input channels to average to form an output channel. If chanbin > 1 then chanaverage is automatically switched to True.
- **timebin** -- Bin width for time averaging. If timebin > 0s then timeaverage is automatically switched to True.
- **omit\_contline\_ms** -- If `True`, don't make the contline ms (\_targets.ms). Only make cont MS (\_targets\_cont.ms). Default is `False`.

### Returns

The results object for the pipeline task is returned.

### Examples

1. Create a science target MS from the corrected column in the input MS:

```
>>> hifv_mstransform()
```

2. Make a phase and bandpass calibrator targets MS from the corrected column in the input MS:

```
>>> hifv_mstransform(intent='PHASE,BANDPASS')
```

## 4.16 pipeline.hifv.cli.hifv\_pbcor

**hifv\_pbcor**(vis=None) → Results

Apply primary beam correction to VLA and VLASS images.

**Parameters**

**vis** -- List of input visibility data.

**Returns**

The results object for the pipeline task is returned.

### Examples

1. Basic pbcor task:

```
>>> hifv_pbcor()
```

## 4.17 pipeline.hifv.cli.hifv\_plotsummary

**hifv\_plotsummary**(vis=None) → ResultsList[Results]

Create pipeline summary plots.

**Parameters**

**vis** -- List of input visibility data.

**Returns**

The results object for the pipeline task is returned.

### Examples

1. Execute the pipeline plotting task:

```
>>> hifv_plotsummary()
```

## 4.18 pipeline.hifv.cli.hifv\_priorcals

**hifv\_priorcals**(vis=None, show\_tec\_maps=None, apply\_tec\_correction=None, apply\_gaincurves=None, apply\_opcal=None, apply\_rqcal=None, apply\_antpos=None, apply\_swpowcal=None, swpow\_spw=None, ant\_pos\_time\_limit=None) → ResultsList[Results]

Runs gaincurves, opacities, requantizer gains, antenna position corrections, tec\_maps, switched power.

**Parameters**

- **vis** (*str*) -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSes, or tar files of MSes. If ASDM files are specified, they will be converted to MS format.

Example: `vis=['X227.ms', 'asdms.tar.gz']`

- **show\_tec\_maps** (*bool*) -- Plot tec maps. Display the plot output from the CASA tec\_maps recipe function.

- **apply\_tec\_correction** -- Apply tec correction. CASA tec\_maps recipe function is executed - this bool determines if gencal is executed and the resulting table applied
- **apply\_gaincurves (bool)** -- Apply gain curves correction, default True.
- **apply\_opcal (bool)** -- Apply opacities correction, default True.
- **apply\_rqcal (bool)** -- Apply requantizer gains correction, default True.
- **apply\_antpos (bool)** -- Apply antenna position corrections, default True.
- **apply\_swpowcal (bool)** -- Apply switched power table, default False. If set True, **apply\_rqcal** is ignored and no requantizer gain correction will be applied.
- **swpow\_spw (str)** -- Spectral-window(s) for plotting: "" ==>all, spw="6, 14"
- **ant\_pos\_time\_limit (int)** -- Antenna position time limit in days, default to 150 days.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Run gaincurves, opacities, requantizer gains and antenna position corrections:

```
>>> hifv_priorcals()
```

## 4.19 pipeline.hifv.cli.hifv\_restoredata

**hifv\_restoredata**(vis=None, session=None, products\_dir=None, copytoraw=None, rawdata\_dir=None, lazy=None, bdfflags=None, ocorr\_mode=None, gainmap=None, asis=None) → Results

Restore flagged and calibration interferometry data from a pipeline run.

**hifv\_restoredata** restores flagged and calibrated data from archived ASDMs and pipeline flagging and calibration data products.

**hifv\_restoredata** assumes that the ASDMs to be restored are present in the directory specified by the **rawdata\_dir** (default: './rawdata').

By default (**copytoraw** = True), **hifv\_restoredata** assumes that for each ASDM in the input list, the corresponding pipeline flagging and calibration data products (in the format produced by the **hifv\_exportdata** task) are present in the directory specified by **products\_dir** (default: '../products'). At the start of the task, these products are copied from the **products\_dir** to the **rawdata\_dir**.

If **copytoraw** = False, **hifv\_restoredata** assumes that these products are to be found in **rawdata\_dir** along with the ASDMs.

The expected flagging and calibration products (for each ASDM) include:

- a compressed tar file of the final flagversions file, e.g. uid\_\_\_\_A002\_X30a93d\_X43e.ms.flagversions.tar.gz
- a text file containing the applycal instructions, e.g. uid\_\_\_\_A002\_X30a93d\_X43e.ms.calapply.txt
- a compressed tar file containing the caltables for the parent session, e.g. uid\_\_\_\_A001\_X74\_X29.session\_3.caltables.tar.gz

**hifv\_restoredata** performs the following operations:

- imports the ASDM(s)
- runs the hanning smoothing task
- removes the default MS.flagversions directory created by the filler
- restores the final MS.flagversions directory stored by the pipeline

- restores the final set of pipeline flags to the MS
- restores the final calibration state of the MS
- restores the final calibration tables for each MS
- applies the calibration tables to each MS

### Parameters

- **vis** -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSes, or tar files of MSes. If ASDM files are specified, they will be converted to MS format.

Example: `vis=['X227.ms', 'asdms.tar.gz']`

- **session** -- List of sessions one per visibility file.

Example: `session=['session_3']`

- **products\_dir** -- Name of the data products directory to copy calibration products from.

Default: `'./products'`

The parameter is effective only when `copytoraw` = True. When `copytoraw` = False, calibration products in `rawdata_dir` will be used.

Example: `products_dir='myproductspath'`

- **copytoraw** -- Copy calibration and flagging tables from `products_dir` to `rawdata_dir` directory.

Default: `True`

Example: `copytoraw=False`.

- **rawdata\_dir** -- Name of the raw data directory. Default: `'./rawdata'`

Example: `rawdata_dir='myrawdatapath'`

- **lazy** -- Use the lazy filler option. Default: False

- **bdfflags** -- Set the BDF flags. Default: False

- **ocorr\_mode** -- Correlation import mode.

Default: `'co'`

- **gainmap** -- If True, map gainfields to a particular list of scans when applying calibration tables.

Default: `False`

- **asis** -- List of tables to import asis.

Default: `'Receiver CalAtmosphere'`

### Returns

The results object for the pipeline task is returned.

### Examples

1. Restore the pipeline results for a single ASDM in a single session:

```
>>> hifv_restoredata (vis=['myVLAsdm'], session=['session_1'], ocorr_mode='ca')
```

## 4.20 pipeline.hifv.cli.hifv\_restorepims

**hifv\_restorepims**(*vis=None, reimaging\_resources=None*) → ResultsList[Results]

Restore VLASS SE per-image MeasurementSet data, resetting flagging, weights, and applying self-calibration.

### Parameters

- **vis** -- List of input visibility data.
- **reimaging\_resources** -- file path of reimaging\_resources.tgz from the SE imaging product.

### Returns

The results object for the pipeline task is returned.

### Examples

1. Basic restorepims task:

```
>>> hifv_restorepims()
```

## 4.21 pipeline.hifv.cli.hifv\_selfcal

**hifv\_selfcal**(*vis=None, refantignore=None, combine=None, selfcalmode=None, refantmode=None, overwrite\_modelcol=None*) → ResultsList[Results]

Perform phase-only self-calibration, per scan row, on VLASS SE images.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **refantignore** -- String list of antennas to ignore.
- **combine** -- Data axes which to combine for solve. Options: "','obs','scan','spw','field', or any comma-separated combination in a single string Example: combine='scan,spw' - Extend solutions over scan boundaries (up to the solint), and combine spws for solving. In selfcalmode='VLASS-SE' use the default value.
- **selfcalmode** -- Heuristics mode selection. Known modes are 'VLASS' and 'VLASS-SE'. Default value is 'VLASS'.
- **refantmode** -- Reference antenna mode.
- **overwrite\_modelcol** -- Always write the model column, even if it already exists.

### Returns

The results object for the pipeline task is returned.

### Examples

1. Basic selfcal task:

```
>>> hifv_selfcal()
```

2. VLASS-SE selfcal usage:

```
>>> hifv_selfcal(selfcalmode='VLASS-SE', combine='field, spw')
```

## 4.22 pipeline.hifv.cli.hifv\_semiFinalBPdcals

**hifv\_semiFinalBPdcals**(*vis=None*, *weakbp=None*, *refantignore=None*, *refant=None*) → ResultsList[Results]

Runs a second delay and bandpass calibration and applies to calibrators to setup for RFI flagging.

### Parameters

- **vis (str, optional)** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **weakbp (Boolean)** -- Activate weak bandpass heuristics. Weak bandpass heuristics on/off - currently not used - see PIPE-104.
- **refantignore (str)** -- String list of antennas to ignore.  
Example: refantignore='ea24,ea15,ea08'
- **refant (str)** -- A csv string of reference antenna(s). When used, disables **refantignore**.  
Example: refant = 'ea01, ea02'

### Returns

The results object for the pipeline task is returned.

### Examples

1. Heuristic flagging:

```
>>> hifv_semiFinalBPdcals()
```

## 4.23 pipeline.hifv.cli.hifv\_solint

**hifv\_solint**(*vis=None*, *limit\_short\_solint=None*, *refantignore=None*, *refant=None*) → ResultsList[Results]

Determines different solution intervals.

The hifv\_solint task determines different solution intervals. Note that the short solint value is switched to 'int' when the minimum solution interval corresponds to one integration.

### Parameters

- **vis (str, optional)** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **limit\_short\_solint (str)** -- Keyword argument in units of seconds to limit the short solution interval. Can be a string or float numerical value in units of seconds of '0.45' or 0.45. Can be set to a string value of 'int'.
- **refantignore (str)** -- String list of antennas to ignore.  
Example: refantignore='ea02, ea03'
- **refant (str)** -- A csv string of reference antenna(s). When used, disables **refantignore**.  
Example: refant = 'ea01, ea02'

### Returns

The results object for the pipeline task is returned.

## Examples

- Determines different solution intervals:

```
>>> hifv_solt()
```

## 4.24 pipeline.hifv.cli.hifv\_statwt

**hifv\_statwt**(vis=None, datacolumn=None, overwrite\_modelcol=None, statwtmode=None) → ResultsList[Results]

Compute statistical weights and write them to measurement set.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **datacolumn** -- Data column used to compute weights. Supported values are "data", "corrected", "residual", and "residual\_data" (case insensitive, minimum match supported).
- **overwrite\_modelcol** -- Always write the model column, even if it already exists.
- **statwtmode** -- Sets the weighting parameters for general VLA ('VLA') or VLASS Single Epoch ('VLASS-SE') use case. Note that the 'VLASS-SE' mode is meant to be used with datacolumn='residual\_data'. Default is 'VLA'.

### Returns

The results object for the pipeline task is returned.

## Examples

- Statistical weighting of the visibilities:

```
>>> hifv_statwt()
```

- Statistical weighting of the visibilities in the Very Large Array Sky Survey Single Epoch use case:

```
>>> hifv_statwt(mode='vllass-se', datacolumn='residual_data')
```

## 4.25 pipeline.hifv.cli.hifv\_syspower

**hifv\_syspower**(vis=None, clip\_sp\_template=None, antexclude=None, apply=None, do\_not\_apply=None) → ResultsList[Results]

Determine amount of gain compression affecting VLA data below Ku-band.

### Parameters

- **vis** -- List of input visibility data.
  - **clip\_sp\_template** -- Acceptable range for Pdiff data; data are clipped outside this range and flagged.
  - **antexclude** -- dictionary in the format of:
- ```
{'L': {'ea02': {'usemedian': True}, 'ea03': {'usemedian': False}}, 'X': {'ea02': {'usemedian': True}, 'ea03': {'usemedian': False}}, 'S': {'ea12': {'usemedian': False}, 'ea22': {'usemedian': False}}}
```

If antexclude is specified with 'usemedian': False, the template values are replaced with 1.0. If 'usemedian': True, the template values are replaced with the median of the good antennas.

- **apply** -- Apply task results to RQ table.
- **do\_not\_apply** -- csv string of band names to not apply.

Example: 'L,X,S'

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Basic syspower task:

```
>>> hifv_sypower()
```

## 4.26 pipeline.hifv.cli.hifv\_testBPdcals

**hifv\_testBPdcals**(vis=None, weakbp=None, refantignore=None, doflagunderspwlimit=None, flagbaddef=None, iglist=None, refant=None) → ResultsList[Results]

Runs initial delay and bandpass calibration to setup for RFI flagging.

#### Parameters

- **vis (str, optional)** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
  - **weakbp (Boolean)** -- Activate weak bandpass heuristics. Weak bandpass heuristics on/off - currently not used - see PIPE-104.
  - **refantignore (str)** -- String list of antennas to ignore.
- Example: refantignore='ea02, ea03'
- **doflagunderspwlimit (Boolean)** -- If the number of bad spws is greater than zero, and the keyword is True, then spws are flagged individually.
  - **flagbaddef (Boolean, optional)** -- Enable/disable bad deformatter flagging. Default is True.
  - **iglist (dict, optional)** -- When flagbaddef is True, skip bad deformatter flagging for elements in the ignore list. Format: {antName:{band:{spw}}} Example: {'ea02': {'L': {0, 1, '10~13'}}}
  - **refant (str)** -- A csv string of reference antenna(s). When used, disables **refantignore**.
- Example: refant = 'ea01, ea02'

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Initial delay calibration to set up heuristic flagging:

```
>>> hifv_testBPdcals()
```

## 4.27 pipeline.hifv.cli.hifv\_vlasetjy

**hifv\_vlasetjy**(*vis=None*, *field=None*, *intent=None*, *spw=None*, *model=None*, *reffile=None*, *fluxdensity=None*, *spix=None*, *reffreq=None*, *scalebychan=None*, *standard=None*) → ResultsList[Results]

Sets flux density scale and fills calibrator model to MeasurementSets.

The hifv\_vlasetjy task does an initial run of setjy on the vis.

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **field** -- List of field names or ids.
- **intent** -- Observing intent of flux calibrators.
- **spw** -- List of spectral window ids.
- **model** -- File location for field model.
- **reffeile** -- Path to file with fluxes for non-solar system calibrators.
- **fluxdensity** -- Specified flux density [I,Q,U,V]; -1 will lookup values.
- **spix** -- Spectral index of fluxdensity. Can be set when fluxdensity is not -1.
- **reffreq** -- Reference frequency for spix. Can be set when fluxdensity is not -1.
- **scalebychan** -- Scale the flux density on a per channel basis or else on a per spw basis.
- **standard** -- Flux density standard.

### Returns

The results object for the pipeline task is returned.

### Examples

- Initial run of setjy:

```
>>> hifv_vlasetjy()
```

## 4.28 pipeline.hifv.cli.hifv\_vlassmasking

**hifv\_vlassmasking**(*vis=None*, *vlass\_ql\_database=None*, *maskingmode=None*, *catalog\_search\_size=None*) → Results

Create clean masks for VLASS Single Epoch (SE) images.

### Parameters

- **vis (str, optional)** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets specified in the hifv\_importdata task.
- **vlass\_ql\_database (str)** -- vlass\_ql\_database - usage in Socorro: /home/vlass/packages/VLASS1Q.fits. Path to a PyBDSF sky catalog in FITS format. Default at AOC is '/home/vlass/packages/VLASS1Q.fits'
- **maskingmode (str)** -- maskingmode options are vlass-se-tier-1 or vlass-se-tier-2. Two modes are: vlass-se-tier-1 (QL mask) and vlass-se-tier-2 (combined mask)
- **catalog\_search\_size (float)** -- catalog\_search\_size in units of degrees. The half-width (in degrees) of the catalog search centered on the image's reference pixel.

### Returns

The results object for the pipeline task is returned.

### Examples

1. Basic vlassmasking task:

```
>>> hifv_vlassmasking()
```

## PIPELINE.HSD.CLI

Single Dish ALMA Tasks

### Functions

|                              |                                                                                  |
|------------------------------|----------------------------------------------------------------------------------|
| <code>hsd_applycal</code>    | Apply the calibration(s) to the data.                                            |
| <code>hsd_atmcor</code>      | Apply offline ATM correction to the data.                                        |
| <code>hsd_baseline</code>    | Detect and validate spectral lines, subtract baseline by masking detected lines. |
| <code>hsd_blfloor</code>     | Flag spectra based on predefined criteria of single dish pipeline.               |
| <code>hsd_exportdata</code>  | Prepare single dish data for export.                                             |
| <code>hsd_flagdata</code>    | Do basic flagging of a list of MeasurementSets.                                  |
| <code>hsd_imaging</code>     | Generate single dish images.                                                     |
| <code>hsd_importdata</code>  | Imports data into the single dish pipeline.                                      |
| <code>hsd_k2jycal</code>     | Derive Kelvin to Jy calibration tables.                                          |
| <code>hsd_restoredata</code> | Restore flagged and calibration single dish data from a pipeline run.            |
| <code>hsd_skycal</code>      | Calibrate data.                                                                  |
| <code>hsd_tsysflag</code>    | Flag deviant system temperature measurements.                                    |

### 5.1 pipeline.hsd.cli.hsd\_applycal

`hsd_applycal(vis: str | list[str] | None = None, field: str | list[str] | None = None, intent: str | list[str] | None = None, spw: str | list[str] | None = None, antenna: str | list[str] | None = None, applymode: str | None = None, flagbackup: bool | None = None, parallel: bool | str | None = None) → ResultsList[SDApplycalResults]`

Apply the calibration(s) to the data.

hsd\_applycal applies the precomputed calibration tables stored in the pipeline context to the set of visibility files using predetermined field and spectral window maps and default values for the interpolation schemes.

#### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets in the pipeline context.  
Example: ['X227.ms']
- **field** -- A string containing the list of field names or field ids to which the calibration will be applied. Defaults to all fields in the pipeline context.  
Example: '3C279', '3C279, M82'
- **intent** -- A string containing the list of intents against which the selected fields will be matched. Defaults to all supported intents in the pipeline context.

Example: '*\*TARGET\**'

- **spw** -- The list of spectral windows and channels to which the calibration will be applied. Defaults to all science windows in the pipeline context.

Example: '17', '11', '15'

- **antenna** -- The selection of antennas to which the calibration will be applied. Defaults to all antennas. Not currently supported.

- **applymode** -- Calibration apply mode.

- 'calflag': calibrate data and apply flags from solutions.
- 'calflagstrict': same as above except flag spws for which calibration is unavailable in one or more tables (instead of allowing them to pass uncalibrated and unflagged). This is the default applymode.
- 'trial': report on flags from solutions, dataset entirely unchanged.
- 'flagonly': apply flags from solutions only, data not calibrated.
- 'flagonlystrict': same as above except flag spws for which calibration is unavailable in one or more tables.
- 'calonly': calibrate data only, flags from solutions NOT applied.

- **flagbackup** -- Backup the flags before the apply.

Default: None (equivalent to True)

- **parallel** -- Execute using CASA HPC functionality, if available. Default is None, which is equivalent to 'automatic' that intends to turn on parallel processing if possible.

Options: 'automatic', 'true', 'false', True, False

### Returns

The results object for the pipeline task is returned.

### Examples

1. Apply the calibration to the target data

```
>>> hsd_applycal(intent='TARGET')
```

2. Specify fields and spectral windows

```
>>> hsd_applycal(field='3C279, M82', spw='17', intent='TARGET')
```

## 5.2 pipeline.hsd.cli.hsd\_atmc

**hsd\_atmc**(atmtype: int | str | List[int] | List[str] | None = None, ditem\_dh: float | str | dict | List[float] | List[str] | List[dict] | None = None, h0: float | str | dict | List[float] | List[str] | List[dict] | None = None, infiles: str | List[str] | None = None, antenna: str | List[str] | None = None, field: str | List[str] | None = None, spw: str | List[str] | None = None, pol: str | List[str] | None = None) → ResultsList[SDATMCorrectionResults]

Apply offline ATM correction to the data.

The hsd\_atmc task provides the capability of offline correction of residual atmospheric features in the calibrated single-dish spectra originated from incomplete calibration mainly due to a difference of elevation angles between ON\_SOURCE and OFF\_SOURCE measurements.

Optimal atmospheric model is automatically determined by default (atmtype = 'auto'). You may specify desired atmospheric model by giving either single integer (apply to all EBs) or a list of integers (models per EB) to atmtype parameter. Please see parameter description for the meanings of integer values.

## Parameters

- **atmtype** -- Type of atmospheric transmission model represented as an integer. Available options are as follows. Integer values can be given as either integer or string, i.e. both 1 and '1' are acceptable.
  - 'auto': perform heuristics to choose best model (default).
  - 1: tropical.
  - 2: mid latitude summer.
  - 3: mid latitude winter.
  - 4: subarctic summer.
  - 5: subarctic winter.

If list of integer is given, it also performs heuristics using the provided values instead of default, [1, 2, 3, 4], which is used when 'auto' is provided. List input should not contain 'auto'.

Default: None (equivalent to 'auto')

- **dtem\_dh** -- Temperature gradient [K/km], e.g. -5.6 ("" = Tool default). The value is directly passed to initialization method for ATM model. Float and string types are acceptable. Float value is interpreted as the value in K/km. String value should be the numeric value with unit such as '-5.6K/km'. When list of values are given, it will trigger heuristics to choose best model from the provided value.

Default: None (equivalent to tool default, -5.6K/km)

- **h0** -- Scale height for water [km], e.g. 2.0 ("" = Tool default). The value is directly passed to initialization method for ATM model. Float and string types are acceptable. Float value is interpreted as the value in kilometer. String value should be the numeric value with unit compatible with length, such as '2km' or '2000m'. When list of values are given, it will trigger heuristics to choose best model from the provided value.

Default: None (equivalent to tool default, 2.0km)

- **infiles** -- ASDM or MS files to be processed. This parameter behaves as data selection parameter. The name specified by infiles must be registered to context before you run hsd\_atmcor.

- **antenna** -- Data selection by antenna names or ids.

Example: 'PM03,PM04', " (all antennas)

- **field** -- Data selection by field names or ids.

Example: '\*Sgr\*,M100', " (all fields)

- **spw** -- Data selection by spw ids.

Example: '3,4' (spw 3 and 4), " (all spws)

- **pol** -- Data selection by polarizations.

Example: 'XX,YY' (correlation XX and YY), " (all polarizations)

## Returns

The results object for the pipeline task is returned.

## Examples

1. Basic usage

```
>>> hsd_atmcor()
```

2. Specify atmospheric model and data selection

```
>>> hsd_atmcor(atmtype=1, antenna='PM03,PM04', field='*Sgr*,M100')
```

3. Specify atmospheric model per EB (atmtype 1 for 1st EB, 2 for 2nd EB)

```
>>> hsd_atmcor(atmtype=[1, 2])
```

## 5.3 pipeline.hsd.cli.hsd\_baseline

**hsd\_baseline**(*fifunc*: *FitFunc* | *None* = *None*, *fitorder*: *FitOrder* | *None* = *None*, *switchpoly*: *bool* | *None* = *None*, *linewindow*: *LineWindow* | *None* = *None*, *linewindowmode*: *str* | *None* = *None*, *edge*: *Tuple[int, int]* | *None* = *None*, *broadline*: *bool* | *None* = *None*, *clusteringalgorithm*: *str* | *None* = *None*, *deviationmask*: *bool* | *None* = *None*, *deviationmask\_sigma\_threshold*: *bool* | *None* = *None*, *parallel*: *str* | *None* = *None*, *infiles*: *List[str]* | *None* = *None*, *field*: *List[str]* | *None* = *None*, *antenna*: *List[str]* | *None* = *None*, *spw*: *List[str]* | *None* = *None*, *pol*: *List[str]* | *None* = *None*) → *SDBaselineResults*

Detect and validate spectral lines, subtract baseline by masking detected lines.

The `hsd_baseline` task subtracts baseline from calibrated spectra. By default, the task tries to find spectral line feature using line detection and validation algorithms. Then, the task puts a mask on detected lines and perform baseline subtraction. The user is able to turn off automatic line masking by setting `linewindow` parameter, which specifies pre-defined line window.

Fitting order is automatically determined by default. It can be disabled by specifying `fitorder` as non-negative value. In this case, the value specified by `fitorder` will be used.

\*WARNING\* Currently, `hsd_baseline` overwrites the result obtained by the previous run. Due to this behavior, users need to be careful about an order of the task execution when they run `hsd_baseline` multiple times with different data selection. Suppose there are two spectral windows (0 and 1) and `hsd_baseline` is executed separately for each `spw` as below,

```
>>> hsd_baseline(spw='0')
>>> hsd_baseline(spw='1')
>>> hsd_blf()
>>> hsd_imaging()
```

Since the second run of `hsd_baseline` overwrites the result for `spw 0` with the data before baseline subtraction, this will not produce correct result for `spw 0`. Proper sequence for this use case is to process each `spw` to the imaging stage separately, which looks like as follows:

```
>>> hsd_baseline(spw='0')
>>> hsd_blf(spw='0')
>>> hsd_imaging(spw='0')
>>> hsd_baseline(spw='1')
>>> hsd_blf(spw='1')
>>> hsd_imaging(spw='1')
```

## Parameters

- **fitfunc** -- Fitting function for baseline subtraction. You can choose either cubic spline ('spline' or 'cspline'), polynomial ('poly' or 'polynomial').

Accepts:

- A string: Applies the same function to all spectral windows (SPWs).
- A dictionary: Maps SPW IDs (int or str) to a specific fitting function.

If an SPW ID is not present in the dictionary, '**cspline**' will be used as the default.

Default: **None** (equivalent to '**cspline**')

- **fitorder** -- Fitting order for polynomial. For cubic spline, it is used to determine how much the spectrum is segmented into.

Accepts:

- An integer: Applies the same order to all SPWs. Valid values: **-1** (automatic), **0**, or any positive integer.
- A dictionary: Maps SPW IDs (int or str) to a specific fitting order.

If an SPW ID is not present in the dictionary, **-1** will be used as the default, triggering automatic order selection.

Default: **None** (equivalent to **-1**)

- **switchpoly** -- Whether to fall back the fits from cubic spline to 1st or 2nd order polynomial when large masks exist at the edges of the spw. Condition for switching is as follows:

- if nmask > nchan/2 => 1st order polynomial
- else if nmask > nchan/4 => 2nd order polynomial
- else => use fitfunc and fitorder

where nmask is a number of channels for mask at edge while nchan is a number of channels of entire spectral window.

Default: **None** (equivalent to **True**)

- **linewindow** -- Pre-defined line window. If this is set, specified line windows are used as a line mask for baseline subtraction instead to determine masks based on line detection and validation stage. Several types of format are acceptable. One is channel-based window.

`[min_chan, max_chan]`

where min\_chan and max\_chan should be an integer. For multiple windows, nested list is also acceptable.

`[[min_chan0, max_chan0], [min_chan1, max_chan1], ...]`

Another way is frequency-based window.

`[min_freq, max_freq]`

where min\_freq and max\_freq should be either a float or a string. If float value is given, it is interpreted as a frequency in Hz. String should be a quantity consisting of "value" and "unit", e.g., '100GHz'. Multiple windows are also supported.

`[[min_freq0, max_freq0], [min_freq1, max_freq1], ...]`

Note that the specified frequencies are assumed to be the value in LSRK frame. Note also that there is a limitation when multiple MSes are processed. If native frequency frame of the data is not LSRK (e.g. TOPO), frequencies need to be converted to that frame. As a result, corresponding channel range may vary between

MSes. However, current implementation is not able to handle such case. Frequencies are converted to desired frame using representative MS (time, position, direction).

In the above cases, specified line windows are applied to all science spws. In case when line windows vary with spw, line windows can be specified by a dictionary whose key is spw id while value is line window. For example, the following dictionary gives different line windows to spws 17 and 19. Other spws, if available, will have an empty line window.

```
{17: [[100, 200], [1200, 1400]], 19: ['112115MHz', '112116MHz']}
```

Furthermore, linewindow accepts MS selection string. The following string gives [[100,200],[1200,1400]] for spw 17 while [1000,1500] for spw 21.

```
"17:100~200;1200~1400,21:1000~1500"
```

The string also accepts frequency with units. Note, however, that frequency reference frame in this case is not fixed to LSRK. Instead, the frame will be taken from the MS (typically TOPO for ALMA). Thus, the following two frequency-based line windows result different channel selections.

```
{19: ['112115MHz', '112116MHz']} # frequency frame is LSRK  
"19:11215MHz~11216MHz" # frequency frame is taken from the data (TOPO for ALMA)
```

None is allowed as a value of dictionary input to indicate that no line detection/validation is required even if manually specified line window does not exist. When None is given as a value and if `linewindowmode` is 'replace', line detection/validation is not performed for the corresponding spw. For example, suppose the following parameters are given for the data with four science spws, 17, 19, 21, and 23.

```
linewindow={17: [112.1e9, 112.2e9], 19: [113.1e9, 113.15e9], 21: None}  
linewindowmode='replace'
```

The task will use given line window for 17 and 19 while the task performs line detection/validation for spw 23 because no line window is set. On the other hand, line detection/validation is skipped for spw 21 due to the effect of None.

Example: [100,200] (channel), [115e9, 115.1e9] (frequency in Hz), ['115GHz', '115.1GHz'] (see above for more examples)

Default: None

- **linewindowmode** -- Merge or replace given manual line window with line detection/validation result. If 'replace' is given, line detection and validation will not be performed. On the other hand, when 'merge' is specified, line detection/validation will be performed and manually specified line windows are added to the result. Note that this has no effect when linewindow for target spw is an empty list. In that case, line detection/validation will be performed regardless of the value of linewindowmode. In case if no linewindow nor line detection/validation are necessary, you should set linewindowmode to 'replace' and specify None as a value of the linewindow dictionary for the spw to apply. See parameter description of `linewindow` for detail.
- **edge** -- Number of edge channels to be dropped from baseline subtraction. The value must be a list with length of 2, whose values specify left and right edge channels, respectively.

Example: [10, 10]

Default: None

- **broadline** -- Try to detect broad component of spectral line if True.

Default: None (equivalent to True)

- **clusteringalgorithm** -- Selection of the algorithm used in the clustering analysis to check the validity of detected line features. The 'kmean' algorithm, hierarchical clustering algorithm, 'hierarchy', and their combination ('both') are so far implemented.  
Default: **None** (equivalent to '**hierarchy**')
- **deviationmask** -- Apply deviation mask in addition to masks determined by the automatic line detection.  
Default: **None** (equivalent to **True**)
- **deviationmask\_sigma\_threshold** -- Threshold factor (F) to detect the deviation. Actual threshold will be median + F \* standard-deviation of the spectrum.  
Default: **None** (equivallent to **5.0**)
- **parallel** -- Execute using CASA HPC functionality, if available.  
Options: '**automatic**', '**true**', '**false**', **True**, **False**.  
Default: **None** (equivalent to '**automatic**').
- **infiles** -- List of data files. These must be a name of MeasurementSets that are registered to context via `hsd_importdata` or `hsd_restoredata` task.  
Example: `vis=['X227.ms', 'X228.ms']` Default: **None** (process all registered MeasurementSets)
- **field** -- Data selection by field.  
Example: '**1**' (select by FIELD\_ID), '**M100\***' (select by field name), '**'**' (all fields)  
Default: **None** (equivalent to '**'**)
- **antenna** -- Data selection by antenna.  
Example: '**1**' (select by ANTENNA\_ID), '**PM03**' (select by antenna name), '**"**' (all antennas)  
Default: **None** (equivalent to '**"**)
- **spw** -- Data selection by spw.  
Example: '**3,4**' (process spw 3 and 4), '**[0,2]**' (spw 0 for first data, 2 for second), '**'**' (all spws)  
Default: **None** (equivalent to '**'**)
- **pol** -- Data selection by polarizations.  
Example: '**0**' (process pol 0), '**[0~1, 0]**' (pol 0 and 1 for first data, only 0 for second), '**'**' (all polarizations)  
Default: **None** (equivalent to '**'**)

## Returns

The results object for the pipeline task is returned.

## Examples

1. Basic usage with automatic line detection and validation

```
>>> hsd_baseline(antenna='PM03', spw='17,19')
```

2. Using pre-defined line windows without automatic line detection and edge channels

```
>>> hsd_baseline(linewindow=[[100, 200], [1200, 1400]], linewindowmode='replace', edge=[10, 10])
```

### 3. Using per spw pre-defined line windows with automatic line detection

```
>>> hsd_baseline(linewindow=[[19: [[390, 550]], 23: [[100, 200], [1200, 1400]]}}, linewindowmode='merge')
```

## 5.4 pipeline.hsd.cli.hsd\_blfag

**hsd\_blfag**(iteration: str | int | None = None, edge: str | int | List[int] | None = None, flag\_tsys: str | bool | None = None, tsys\_thresh: str | Integral | None = None, flag\_pfr: str | bool | None = None, pfr\_thresh: str | Integral | None = None, flag\_pofr: str | bool | None = None, pofr\_thresh: str | Integral | None = None, pfr\_thresh: str | Integral | None = None, flag\_pofr: str | bool | None = None, pofr\_thresh: str | Integral | None = None, flag\_pfrm: str | bool | None = None, pfrm\_thresh: str | Integral | None = None, pfrm\_nmean: str | Integral | None = None, flag\_pofrm: str | bool | None = None, pofrm\_thresh: str | Integral | None = None, pofrm\_nmean: str | Integral | None = None, plotflag: str | bool | None = None, parallel: str | bool | None = None, infiles: str | List[str] | None = None, antenna: str | List[str] | None = None, field: str | List[str] | None = None, spw: str | List[str] | None = None, pol: str | List[str] | None = None) → ResultsList[Results]

Flag spectra based on predefined criteria of single dish pipeline.

Data are flagged based on several flagging rules. Available rules are: expected rms, calculated rms, and running mean of both pre-fit and post-fit spectra. Tsys flagging is also available.

In addition, the heuristics script creates many plots for each stage. Those plots are included in the weblog.

### Parameters

- **iteration** -- Number of iterations to perform sigma clipping to calculate threshold value of flagging.  
Default: None (equivalent to 5.0)
- **edge** -- Number of channels to be dropped from the edge. The value must be a list of integer with length of one or two. If list length is one, same number will be applied both side of the band.  
Example: [10,20], [10]  
Default: None (equivalent to [0, 0])
- **flag\_tsys** -- Activate (True) or deactivate (False) Tsys flag. Default is None which is equivalent to True.
- **tsys\_thresh** -- Threshold value for Tsys flag. Default is None which sets 3.0 as a threshold.
- **flag\_pfr** -- Activate (True) or deactivate (False) flag by expected rms of pre-fit spectra. Default is None which is equivalent to True.
- **pfr\_thresh** -- Threshold value for flag by expected rms of pre-fit spectra. Default is None which sets 3.0 to a threshold.
- **flag\_pofr** -- Activate (True) or deactivate (False) flag by expected rms of post-fit spectra. Default is None which is equivalent to True.
- **pofr\_thresh** -- Threshold value for flag by expected rms of post-fit spectra. Default is None which sets 1.333 to a threshold.
- **flag\_pfrm** -- Activate (True) or deactivate (False) flag by rms of pre-fit spectra. Default is None which is equivalent to True.
- **pfrm\_thresh** -- Threshold value for flag by rms of pre-fit spectra. Default is None which sets 4.5 to a threshold.
- **flag\_pofrm** -- Activate (True) or deactivate (False) flag by rms of post-fit spectra. Default is None which is equivalent to True.

- **pofr\_thresh** -- Threshold value for flag by rms of post-fit spectra. Default is None which sets 4.0 to a threshold.
- **flag\_prfrm** -- Activate (True) or deactivate (False) flag by running mean of pre-fit spectra. Default is None which is equivalent to True.
- **prfrm\_thresh** -- Threshold value for flag by running mean of pre-fit spectra. Default is None which sets 5.5 to a threshold.
- **prfrm\_nmean** -- Number of channels for running mean of pre-fit spectra. Default is None which sets 5 channels for running mean.
- **flag\_pofrm** -- Activate (True) or deactivate (False) flag by running mean of post-fit spectra. Default is None which is equivalent to True.
- **pofrm\_thresh** -- Threshold value for flag by running mean of post-fit spectra. Default is None which sets 5.0 to a threshold.
- **pofrm\_nmean** -- Number of channels for running mean of post-fit spectra. Default is None which sets 5 channels for running mean.
- **plotflag** -- True to plot result of data flagging. Default is None which is equivalent to True.
- **parallel** -- Execute using CASA HPC functionality, if available.  
Options: 'automatic', 'true', 'false', True, False  
Default: None (equivalent to 'automatic')
- **infiles** -- ASDM or MS files to be processed. This parameter behaves as data selection parameter. The name specified by infilenames must be registered to context before you run hsd\_bfflag.  
Default: None (process all registered data)
- **antenna** -- Data selection by antenna names or ids.  
Example: 'PM03,PM04', " (all antennas)  
Default: None (process all antennas)
- **field** -- Data selection by field names or ids.  
Example: '\*Sgr\*,M100', " (all fields)  
Default: None (process all science fields)
- **spw** -- Data selection by spw ids.  
Example: '3,4' (spw 3 and 4), " (all spws)  
Default: None (process all science spws)
- **pol** -- Data selection by polarizations.  
Example: 'XX,YY' (correlation XX and YY), " (all polarizations)  
Default: None (process all polarizations)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. flagging with all rules

```
>>> hsd_blf()
```

## 5.5 pipeline.hsd.cli.hsd\_exportdata

**hsd\_exportdata**(*pprfile*: *list[str] = None*, *targetimages*: *list[str] = None*, *products\_dir*: *str = None*) → Results

Prepare single dish data for export.

The hsd\_exportdata task exports the data defined in the pipeline context and exports it to the data products directory, converting and or packing it as necessary.

The current version of the task exports the following products

- a FITS image for each selected science target source image
- a tar file per ASDM containing the final flags version and blparam
- a tar file containing the file web log

### Parameters

- **pprfile** -- Name of the pipeline processing request to be exported. Defaults to a file matching the template 'PPR\_\*.xml'.

Example: `pprfile=['PPR_GRB021004.xml']`

- **targetimages** -- List of science target images to be exported. Defaults to all science target images recorded in the pipeline context.

Example: `targetimages=['NGC3256.band3', 'NGC3256.band6'], targetimages=['r_aqr.CM02.spw5.line0.XXYY.sd.im', 'r_aqr.CM02.spw5.XXYY.sd.cont.im']`

- **products\_dir** -- Name of the data products subdirectory. Defaults to '.'.

Example: `products_dir='..../products'`

### Returns

The results object for the pipeline task is returned.

### Examples

1. Export the pipeline results for a single session to the data products directory

```
>>> !mkdir ..../products
>>> hsd_exportdata (products_dir='..../products')
```

## 5.6 pipeline.hsd.cli.hsd\_flagdata

**hsd\_flagdata**(*vis*: *List[str] | None = None*, *autocorr*: *str | bool | None = None*, *shadow*: *str | bool | None = None*, *scan*: *str | bool | None = None*, *scannumber*: *str | None = None*, *intents*: *str | None = None*, *edgespw*: *str | bool | None = None*, *fracspw*: *str | None = None*, *fracspwfps*: *str | float | None = None*, *online*: *str | bool | None = None*, *fileonline*: *str | None = None*, *template*: *str | bool | None = None*, *filemplate*: *str | None = None*, *pointing*: *str | bool | None = None*, *filepointing*: *str | None = None*, *incompleteraster*: *str | bool | None = None*, *hm\_tbuff*: *str | None = None*, *tbuff*: *str | float | None = None*, *qa0*: *str | bool | None = None*, *qa2*: *str | bool | None = None*, *parallel*: *str | bool | None = None*, *flagbackup*: *str | bool | None = None*) → ResultsList[Results]

Do basic flagging of a list of MeasurementSets.

The hsd\_flagdata data performs basic flagging operations on a list of MeasurementSets including:

- applying online flags
- applying a flagging template
- shadowed antenna data flagging
- scan-based flagging by intent or scan number
- edge channel flagging

### Parameters

- **vis** -- The list of input MeasurementSets. Defaults to the list of MeasurementSets defined in the pipeline context.
- **autocorr** -- Flag autocorrelation data.  
Default: None (equivalent to False)
- **shadow** -- Flag shadowed antennas.  
Default: None (equivalent to True)
- **scan** -- Flag a list of scans and intents specified by scannumber and intents.  
Default: None (equivalent to True)
- **scannumber** -- A string containing a comma delimited list of scans to be flagged.  
Default: None (equivalent to "")
- **intents** -- A string containing a comma delimited list of intents against which the scans to be flagged are matched. Defaults to intents that are not relevant to pipeline processing.  
Example: '\*BANDPASS\*'
- **edgespw** -- Flag the edge spectral window channels.  
Default: None (equivalent to True)
- **fracspw** -- Fraction of the baseline correlator TDM edge channels to be flagged.  
Default: None (equivalent to 0.03125)
- **fracspwfps** -- Fraction of the ACA correlator TDM edge channels to be flagged.  
Default: None (equivalent to 0.048387)
- **online** -- Apply the online flags.  
Default: None (equivalent to True)
- **fileonline** -- File containing the online flags. These are computed by the h\_init or hsd\_importdata data tasks. If the online flags files are undefined a name of the form 'msname.flagonline.txt' is assumed.
- **template** -- Apply a flagging template.  
Default: None (equivalent to True)
- **filitemap** -- The name of a text file that contains the flagging template for RFI, birdies, telluric lines, etc. If the template flags files is undefined a name of the form 'msname.flagtemplate.txt' is assumed.
- **pointing** -- Apply a flagging template for pointing flag.  
Default: None (equivalent to True)
- **filepointing** -- The name of a text file that contains the flagging template for pointing flag. If the template flags files is undefined a name of the form 'msname.flagpointing.txt' is assumed.

- **incompleteraster** -- Apply commands to flag incomplete raster sequence. If this is False, relevant commands in filepointing are simply commented out.  
Default: None (equivalent to True)
- **hm\_tbuff** -- The heuristic for computing the default time interval padding parameter. The options are 'halfint' and 'manual'. In 'halfint' mode tbuff is set to half the maximum of the median integration time of the science and calibrator target observations.  
Default: None (equivalent to 'halfint')
- **tbuff** -- The time in seconds used to pad flagging command time intervals if hm\_tbuff='manual'.  
Default: None (equivalent to 0.0)
- **qa0** -- QA0 flags
- **qa2** -- QA2 flags
- **parallel** -- Execute using CASA HPC functionality, if available.  
Options: 'automatic', 'true', 'false', True, False  
Default: None (equivalent to 'automatic')
- **flagbackup** -- Back up any pre-existing flags before applying new ones.  
Default: None (equivalent to True)

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Do basic flagging on a MeasurementSet

```
>>> hsd_flagdata()
```

2. Do basic flagging on a MeasurementSet flagging additional scans selected by number as well.

```
>>> hsd_flagdata(scannumber='13,18')
```

## 5.7 pipeline.hsd.cli.hsd\_imaging

**hsd\_imaging**(mode: str | None = None, restfreq: str | None = None, infiles: List[str] | None = None, field: str | None = None, spw: str | None = None) → SDImagingResults

Generate single dish images.

The hsd\_imaging task generates single dish images per antenna as well as combined image over whole antennas for each field and spectral window. Image configuration (grid size, number of pixels, etc.) is automatically determined based on meta data such as antenna diameter, map extent, etc.

Generated images are either in REST frame (ephemeris sources) or in LSRK frame (others).

**Parameters**

- **mode** -- Imaging mode controls imaging parameters in the task. Accepts either "line" (spectral line imaging) or "ampcal" (image settings for amplitude calibrator).  
Default: **None** (equivalent to '**line**')
- **restfreq** -- Rest frequency. Defaults to None, it executes without rest frequency.

- **infiles** -- List of data files. These must be a name of MeasurementSets that are registered to context via hsd\_importdata or hsd\_restoredata tasks.

Example: `vis=['uid__A002_X85c183_X36f.ms', 'uid__A002_X85c183_X60b.ms']`

Default: `None` (process all registered MeasurementSets)

- **field** -- Data selection by field names or ids.

Example: `"*Sgr*,M100"`

Default: `None` (process all science fields)

- **spw** -- Data selection by spw ids.

Example: `"3,4"` (generate images for spw 3 and 4)

Default: `None` (process all science spws)

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Generate images with default settings and context

```
>>> hsd_imaging()
```

2. Generate images with amplitude calibrator and specific parameters

```
>>> hsd_imaging(mode='ampcal', field='*Sgr*,M100', spw='17,19')
```

## 5.8 pipeline.hsd.cli.hsd\_importdata

**hsd\_importdata**(`vis: List[str] | None = None, session: List[str] | None = None, hm_rasterscan: str | None = None, parallel: str | bool | None = None, asis: str | None = None, process_caldevice: bool | None = None, overwrite: bool | None = None, nocopy: bool | None = None, bdfflags: bool | None = None, datacolumns: Dict | None = None, lazy: bool | None = None, with_pointing_correction: bool | None = None, createmms: str | None = None`) → ResultsList[Results]

Imports data into the single dish pipeline.

The hsd\_importdata task loads the specified visibility data into the pipeline context unpacking and / or converting it as necessary.

If the `overwrite` input parameter is set to False and the task is asked to convert an input ASDM input to an MS, then when the output MS already exists in the output directory, the importasdm conversion step is skipped, and the existing MS will be imported instead.

#### Parameters

- **vis** -- List of visibility data files. These may be ASDMs, tar files of ASDMs, MSes, or tar files of MSes, If ASDM files are specified, they will be converted to MS format.

Example: `vis=['X227.ms', 'asdms.tar.gz']`

- **session** -- List of sessions to which the visibility files belong. Defaults to a single session containing all the visibility files, otherwise a session must be assigned to each vis file.

Example: `session=['Session_1', 'Sessions_2']`

- **hm\_rasterscan** -- Heuristics method for raster scan analysis. Two analysis modes, time-domain analysis ('time') and direction analysis ('direction'), are available.

Default: `None` (equivalent to '`time`')

- **parallel** -- Execute using CASA HPC functionality, if available.

Options: `'automatic'`, `'true'`, `'false'`, `True`, `False`

Default: `None` (equivalent to '`automatic`')

- **asis** -- Creates verbatim copies of the ASDM tables in the output MS. The value given to this option must be a list of table names separated by space characters. Default value, `None`, is equivalent to the following list.

```
'SBSummary ExecBlock Annotation Antenna Station Receiver Source CalAtmosphere
CalWVR SpectralWindow'
```

Example: `'Receiver', ''`

- **process\_caldevice** -- Import the ASDM caldevice table.

Example: `True`

Default: `None` (equivalent to `True`)

- **overwrite** -- Overwrite existing files on import. When converting ASDM to MS, if `overwrite=False` and the MS already exists in output directory, then this existing MS dataset will be used instead.

Default: `None` (equivalent to `False`)

- **nocopy** -- Disable copying of MS to working directory

- **bdfflags** -- Apply BDF flags on import.

- **datacolumns** -- Dictionary defining the data types of existing columns. The format is:

```
{'data': 'data type 1'}
```

or

```
{'data': 'data type 1', 'corrected': 'data type 2'}
```

For ASDMs the data type can only be RAW and one can only specify it for the data column. For MSes one can define two different data types for the DATA and CORRECTED\_DATA columns and they can be any of the known data types (RAW, REGCAL\_CONTLINE\_ALL, REGCAL\_CONTLINE\_SCIENCE, SELFCAL\_CONTLINE\_SCIENCE, REGCAL\_LINE\_SCIENCE, SELFCAL\_LINE\_SCIENCE, BASELINED, ATMCORR). The intent selection strings \_ALL or \_SCIENCE can be skipped. In that case the task determines this automatically by inspecting the existing intents in the dataset. Usually, a single `datacolumns` dictionary is used for all datasets. If necessary, one can define a list of dictionaries, one for each EB, with different setups per EB. If no type is specified, `{'data':'raw'}` will be assumed.

- **lazy** -- Use the lazy filter import.

Default: `None` (equivalent to `False`)

- **with\_pointing\_correction** -- Apply pointing correction to DIRECTION. Add (ASDM::Pointing::encoder - ASDM::Pointing::pointingDirection) to the value to be written in MS::Pointing::direction.

Default: `None` (equivalent to `True`)

- **createmms** -- Create an MMS.

Default: `None` (equivalent to `False`)

## Returns

The results object for the pipeline task is returned.

## Examples

- Load an ASDM list in the ./rawdata subdirectory into the context.

```
>>> hsd_importdata (vis=['../rawdata/uid__A002_X30a93d_X43e', '../rawdata/uid_A002_x30a93d_X44e
˓→'])
```

- Load an MS in the current directory into the context.

```
>>> hsd_importdata (vis=['uid__A002_X30a93d_X43e.ms'])
```

- Load a tarred ASDM in ./rawdata into the context.

```
>>> hsd_importdata (vis=['../rawdata/uid__A002_X30a93d_X43e.tar.gz'])
```

- Import a list of MeasurementSets.

```
>>> myvislist = ['uid__A002_X30a93d_X43e.ms', 'uid_A002_x30a93d_X44e.ms']
>>> hsd_importdata(vis=myvislist)
```

## 5.9 pipeline.hsd.cli.hsd\_k2jycal

**hsd\_k2jycal**(*dbservice: bool | None = None*, *endpoint: str | None = None*, *reffile: str | None = None*, *infiles: str | List[str] | None = None*, *caltable: str | List[str] | None = None*) → ResultsList[SDK2JyCalResults]

Derive Kelvin to Jy calibration tables.

Derive the Kelvin to Jy calibration for list of MeasurementSets.

### Parameters

- **dbservice** -- Whether or not accessing Jy/K DB to retrieve conversion factors.

Default: None (equivalent to True)

- **endpoint** -- Which endpoints to use for query.

Options: 'asdm', 'model-fit', 'interpolation'

Default: None (equivalent to 'asdm')

- **reffile** -- Path to a file containing Jy/K factors for science data, which must be provided by associating calibrator reduction or the observatory measurements. Jy/K factor must take into account all efficiencies, i.e., it must be a direct conversion factor from Ta\* to Jy. The file must be in either MS-based or session-based format. The MS-based format must be in an CSV format with five fields: MS name, antenna name, spectral window id, polarization string, and Jy/K conversion factor. Example for the file is as follows:

```
MS,Antenna,Spwid,Polarization,Factor
uid__A002_X316307_X6f.ms,CM03,5,XX,10.0
uid__A002_X316307_X6f.ms,CM03,5,YY,12.0
uid__A002_X316307_X6f.ms,PM04,5,XX,2.0
uid__A002_X316307_X6f.ms,PM04,5,YY,5.0
```

The first line in the above example is a header which may or may not exist. Example for the session-based format is as follows:

```
#OUSID=XXXXXX
#OBJECT=Uranus
#FLUXJY=yy,zz,aa
#FLUXFREQ=YY,ZZ,AA
#sessionID,ObservationStartDate(UTC),ObservationEndDate(UTC),
Antenna,BandCenter(MHz),BandWidth(MHz),POL,Factor
1,2011-11-11 01:00:00,2011-11-11 01:30:00,CM02,86243.0,500.0,I,10.0
1,2011-11-11 01:00:00,2011-11-11 01:30:00,CM02,86243.0,1000.0,I,30.0
1,2011-11-11 01:00:00,2011-11-11 01:30:00,CM03,86243.0,500.0,I,50.0
1,2011-11-11 01:00:00,2011-11-11 01:30:00,CM03,86243.0,1000.0,I,70.0
1,2011-11-11 01:00:00,2011-11-11 01:30:00,ANONYMOUS,86243.0,500.0,I,30.0
1,2011-11-11 01:00:00,2011-11-11 01:30:00,ANONYMOUS,86243.0,1000.0,I,50.0
2,2011-11-13 01:45:00,2011-11-13 02:15:00,PM04,86243.0,500.0,I,90.0
2,2011-11-13 01:45:00,2011-11-13 02:15:00,PM04,86243.0,1000.0,I,110.0
2,2011-11-13 01:45:00,2011-11-13 02:15:00,ANONYMOUS,86243.0,500.0,I,90.0
2,2011-11-13 01:45:00,2011-11-13 02:15:00,ANONYMOUS,86243.0,1000.0,I,110.0
```

Lines starting with '#' are meta data and header. The header must exist. The factor to apply is identified by matching the session ID, antenna name, frequency and polarization of data in each line of the file. Note the observation date is supplementary information and not used for the matching so far. The lines whose antenna name is 'ANONYMOUS' are used when there is no measurement for specific antenna in the session. In the above example, if science observation of session 1 contains the antenna PM04, Jy/K factor for ANONYMOUS antenna will be applied since there is no measurement for PM04 in session 1. If no file name is specified or specified file doesn't exist, all Jy/K factors are set to 1.0.

Example: refile="", refile='working/jyperk.csv'

- **infiles** -- List of input MeasurementSets.

Example: vis='ngc5921.ms'

- **caltable** -- Name of output gain calibration tables. Name is automatically created from infiles if None is given.

Example: caltable='ngc5921.gcal'

## Returns

The results object for the pipeline task is returned.

## Examples

1. Compute the Kevin to Jy calibration tables for a list of MeasurementSets:

```
>>> hsd_k2jycal()
```

## 5.10 pipeline.hsd.cli.hsd\_restoredata

```
hsd_restoredata(vis: List[str] = None, session: str | None = None, products_dir: str | None = None, copytoraw: bool | None = None, rawdata_dir: str | None = None, lazy: bool | None = None, bdfflags: bool | None = None, ocorr_mode: str | None = None, asis: str | None = None, hm_rasterscan: str | None = None) → Results
```

Restore flagged and calibration single dish data from a pipeline run.

The hsd\_restoredata task restores flagged and calibrated MeasurementSets from archived ASDMs and pipeline flagging and calibration data products.

`hsd_restoredata` assumes that the ASDMs to be restored are present in the directory specified by the `rawdata_dir` (default: `'./rawdata'`).

By default (`copytoraw` = `True`), `hsd_restoredata` assumes that for each ASDM in the input list, the corresponding pipeline flagging and calibration data products (in the format produced by the `hsd_exportdata` task) are present in the directory specified by `products_dir` (default: `'../products'`). At the start of the task, these products are copied from the `products_dir` to the `rawdata_dir`.

If `copytoraw` = `False`, `hsd_restoredata` assumes that these products are to be found in `rawdata_dir` along with the ASDMs.

The expected flagging and calibration products (for each ASDM) include:

- a compressed tar file of the final flagversions file, e.g. `uid____A002_X30a93d_X43e.ms.flagversions.tar.gz`
- a text file containing the applycal instructions, e.g. `uid____A002_X30a93d_X43e.ms.calapply.txt`
- a compressed tar file containing the caltables for the parent session, e.g. `uid____A001_X74_X29.session_3.caltables.tar.gz`

`hsd_restoredata` performs the following operations:

- imports the ASDM(s)
- removes the default MS.flagversions directory created by the filler
- restores the final MS.flagversions directory stored by the pipeline
- restores the final set of pipeline flags to the MS
- restores the final calibration state of the MS
- restores the final calibration tables for each MS
- applies the calibration tables to each MS

When importing the ASDM and converting it to a MeasurementSet (MS), if the output MS already exists in the output directory, then the importasdm conversion step is skipped, and the existing MS will be imported instead.

## Parameters

- **`vis`** -- List of raw visibility data files to be restored. Assumed to be in the directory specified by `rawdata_dir`.  
Example: `vis=['uid____A002_X30a93d_X43e']`
- **`session`** -- List of sessions one per visibility file.  
Example: `session=['session_3']`
- **`products_dir`** -- Name of the data products directory to copy calibration products from. The parameter is effective only when `copytoraw=True`. When `copytoraw=False`, calibration products in `rawdata_dir` will be used.  
Example: `products_dir='myproductspath'`  
Default: `None` (equivalent to `'../products'`)
- **`copytoraw`** -- Copy calibration and flagging tables from `products_dir` to `rawdata_dir` directory.  
Example: `copytoraw=False`  
Default: `None` (equivalent to `True`)
- **`rawdata_dir`** -- Name of the raw data directory.  
Example: `rawdata_dir='myrawdatopath'`  
Default: `None` (equivalent to `'./rawdata'`)

- **lazy** -- Use the lazy filler option.  
Example: `lazy=True`  
Default: `None` (equivalent to `False`)
- **bdfflags** -- Apply BDF flags on import.  
Example: `bdfflags=False`  
Default: `None` (equivalent to `True`)
- **ocorr\_mode** -- Selection of baseline correlation to import. Valid only if input visibility is ASDM. See a document of CASA, casatasks:importasdm, for available options.  
Example: `ocorr_mode='ca'`  
Default: `None` (equivalent to `'ao'`)
- **asis** -- Creates verbatim copies of the ASDM tables in the output MS. The value given to this option must be a list of table names separated by space characters. Default value, `None`, is equivalent to the following list.

`'SBSummary ExecBlock Annotation Antenna Station Receiver Source CalAtmosphere  
CalWVR SpectralWindow'`

- Example: `asis='Source Receiver'`
- **hm\_rasterscan** -- Heuristics method for raster scan analysis. Two analysis modes, time-domain analysis ('time') and direction analysis ('direction'), are available.  
Default: `None` (equivalent to `'time'`)

#### Returns

The results object for the pipeline task is returned.

#### Examples

1. Restore the pipeline results for a single ASDM in a single session

```
>>> hsd_restoredata (vis=['uid__A002_X30a93d_X43e'], session=['session_1'], ocorr_mode='ao')
```

## 5.11 pipeline.hsd.cli.hsd\_skycal

**hsd\_skycal**(calmode: str | None = `None`, fraction: float | None = `None`, noff: int | None = `None`, width: float | None = `None`, elongated: bool | None = `None`, parallel: bool | str | None = `None`, infiles: str | None = `None`, field: str | None = `None`, spw: str | None = `None`, scan: str | None = `None`) → ResultsList[SDSSkyCalResults]

Calibrate data.

The `hsd_skycal` generates a caltable for sky calibration that stores reference spectra, which is to be subtracted from on-source spectra to filter out non-source contribution.

#### Parameters

- **calmode** -- Calibration mode. Available options are 'auto' (default), 'ps', 'otf', and 'otfraster'. When 'auto' is set, the task will use preset calibration mode that is determined by inspecting data. 'ps' mode is simple position switching using explicit reference scans. Other two modes, 'otf' and 'otfraster', will generate reference data from scans at the edge of the map. Those modes are intended for OTF observation and the former is defined for generic scanning pattern such as Lissajous, while the latter is specific use for raster scan.

Options: `'auto', 'ps', 'otf', 'otfraster'`

Default: `None` (equivalent to `'auto'`)

- **fraction** -- Sub-parameter for calmode. Edge marking parameter for 'otf' and 'otfraster' mode. It specifies a number of OFF scans as a fraction of total number of data points.

**Options:** String style like '20%', or float value less than 1.0.

For 'otfraster' mode, you can also specify 'auto'.

Default: **None** (equivalent to '10%')

- **noff** -- Sub-parameter for calmode. Edge marking parameter for 'otfraster' mode. It is used to specify a number of OFF scans near edge directly instead to specify it by fractional number by 'fraction'. If it is set, the value will come before setting by 'fraction'.

Options: any positive integer value

Default: **None** (equivalent to '')

- **width** -- Sub-parameter for calmode. Edge marking parameter for 'otf' mode. It specifies pixel width with respect to a median spatial separation between neighboring two data in time. Default will be fine in most cases.

Options: any float value

Default: **None** (equivalent to 0.5)

- **elongated** -- Sub-parameter for calmode. Edge marking parameter for 'otf' mode. Please set True only if observed area is elongated in one direction.

Default: **None** (equivalent to False)

- **parallel** -- Execute using CASA HPC functionality, if available.

Options: 'automatic', 'true', 'false', True, False

Default: **None** (equivalent to 'automatic')

- **infiles** -- List of data files. These must be a name of MeasurementSets that are registered to context via hsd\_importdata or hsd\_restoredata task.

Example: vis=['X227.ms', 'X228.ms']

- **field** -- Data selection by field name.

- **spw** -- Data selection by spw.

Example: '3,4' (generate caltable for spw 3 and 4), ['0','2'] (spw 0 for first data, 2 for second)

Default: **None** (process all science spws)

- **scan** -- Data selection by scan number. (default all scans)

Example: '22,23' (use scan 22 and 23 for calibration), ['22','24'] (scan 22 for first data, 24 for second)

Default: None (process all scans)

## Returns

The results object for the pipeline task is returned.

## Examples

1. Generate caltables for all data managed by context.

```
>>> default(hsd_skycal)
>>> hsd_skycal()
```

## 5.12 pipeline.hsd.cli.hsd\_tsysflag

```
hsd_tsysflag(vis: List[str] | None = None, caltable: List[str] | None = None, flag_nmedian: str | bool | None = None, fnm_limit: str | Integral | None = None, fnm_byfield: str | bool | None = None, flag_derivative: str | bool | None = None, fd_max_limit: str | Integral | None = None, flag_edgechans: str | bool | None = None, fe_edge_limit: str | Integral | None = None, flag_fieldshape: str | bool | None = None, ff_refintent: str | None = None, ff_max_limit: str | Integral | None = None, flag_birdies: str | bool | None = None, fb_sharps_limit: str | Integral | None = None, flag_toomany: str | bool | None = None, tmf1_limit: str | Integral | None = None, tmeff1_limit: str | Integral | None = None, metric_order: str | None = None, normalize_tsys: str | bool | None = None, filetemplate: str | None = None) → ResultsList[Results]
```

Flag deviant system temperature measurements.

Flag deviant system temperature measurements for single dish measurements. This is done by running a sequence of flagging sub-tasks (tests), each looking for a different type of possible error.

If a file with manual Ts sys flags is provided with the 'filetemplate' parameter, then these flags are applied prior to the evaluation of the flagging heuristics listed below.

The tests are:

1. Flag Ts sys spectra with high median values
2. Flag Ts sys spectra with high median derivatives. This is meant to spot spectra that are 'ringing'.
3. Flag the edge channels of the Ts sys spectra in each SpW.
4. Flag Ts sys spectra whose shape is different from that associated with the BANDPASS intent.
5. Flag 'birdies'.
6. Flag the Ts sys spectra of all antennas in a timestamp and spw if

proportion of antennas already flagged in this timestamp and spw exceeds a threshold, and flag Ts sys spectra for all antennas and all timestamps in a spw, if proportion of antennas that are already entirely flagged in all timestamps exceeds a threshold.

### Parameters

- **vis** -- List of input MeasurementSets (Not used)
- **caltable** -- List of input Ts sys calibration tables.  
Example: caltable=['X132.ms.tsys.s2.tbl']  
Default: None (equivalent to []) - Use the table currently stored in the pipeline context)
- **flag\_nmedian** -- True to flag Ts sys spectra with high median value.  
Default: None (equivalent to True)
- **fm\_limit** -- Flag spectra with median value higher than fnm\_limit \* median of this measure over all spectra.  
Default: None (equivalent to 2.0)
- **fm\_byfield** -- Evaluate the nmedian metric separately for each field.  
Default: None (equivalent to True)

- **flag\_derivative** -- True to flag Tsys spectra with high median derivative.  
Default: None (equivalent to True)
- **fd\_max\_limit** -- Flag spectra with median derivative higher than fd\_max\_limit \* median of this measure over all spectra.  
Default: None (equivalent to 5.0)
- **flag\_edgechans** -- True to flag edges of Tsys spectra.  
Default: None (equivalent to True)
- **fe\_edge\_limit** -- Flag channels whose channel to channel difference > fe\_edge\_limit \* median across spectrum.  
Default: None (equivalent to 3.0)
- **flag\_fieldshape** -- True to flag Tsys spectra with a radically different shape to those of the ff\_refintent.  
Default: None (equivalent to True)
- **ff\_refintent** -- Data intent that provides the reference shape for 'flag\_fieldshape'.  
Default: None (equivalent to 'BANDPASS')
- **ff\_max\_limit** -- Flag Tsys spectra with 'fieldshape' metric values > ff\_max\_limit.  
Default: None (equivalent to 13)
- **flag\_birdies** -- True to flag channels covering sharp spectral features.  
Default: None (equivalent to True)
- **fb\_sharps\_limit** -- Flag channels bracketing a channel to channel difference > fb\_sharps\_limit.  
Default: None (equivalent to 0.15)
- **flag\_toomany** -- True to flag Tsys spectra for which a proportion of antennas for given timestamp and/or proportion of antennas that are entirely flagged in all timestamps exceeds their respective thresholds.  
Default: None (equivalent to True)
- **tmf1\_limit** -- Flag Tsys spectra for all antennas in a timestamp and spw if proportion of antennas already flagged in this timestamp and spw exceeds tmf1\_limit.  
Default: None (equivalent to 0.666)
- **tmeff1\_limit** -- Flag Tsys spectra for all antennas and all timestamps in a spw, if proportion of antennas that are already entirely flagged in all timestamps exceeds tmeff1\_limit.  
Default: None (equivalent to 0.666)
- **metric\_order** -- Order in which to evaluate the flagging metrics that are enabled. Disabled metrics are skipped. Default order is as follows:  
nmedian derivative edgechans fieldshape birdies toomany
- **normalize\_tsys** -- True to create a normalized Tsys table that is used to evaluate the Tsys flagging metrics. All newly found flags are also applied to the original Tsys caltable that continues to be used for subsequent calibration.  
Default: None (equivalent to False)
- **filetemplate** -- The name of a text file that contains the manual Tsys flagging template. If the template flags file is undefined, a name of the form 'msname.flagstemplate.txt' is assumed.

**Returns**

The results object for the pipeline task is returned.

**Examples**

1. Flag Tsys measurements using currently recommended tests:

```
>>> hsd_tsysflag()
```

2. Flag Tsys measurements using all recommended tests apart from that using the 'fieldshape' metric:

```
>>> hsd_tsysflag(flag_fieldshape=False)
```

**PIPELINE.HSDN.CLI**

Single Dish Nobeyama Tasks

## Functions

|                               |                                                                       |
|-------------------------------|-----------------------------------------------------------------------|
| <code>hsdn_exportdata</code>  | Prepare single dish data for export.                                  |
| <code>hsdn_importdata</code>  | Imports Nobeyama data into the single dish pipeline.                  |
| <code>hsdn_restoredata</code> | Restore flagged and calibration single dish data from a pipeline run. |

## 6.1 pipeline.hsdn.cli.hsdn\_exportdata

`hsdn_exportdata(pprfile: list[str] = None, targetimages: list[str] = None, products_dir: str = None) → Results`

Prepare single dish data for export.

The hsdn\_exportdata task exports the data defined in the pipeline context and exports it to the data products directory, converting and or packing it as necessary.

The current version of the task exports the following products

- a FITS image for each selected science target source image
- a tar file per MS containing the final flags version and bparam
- a tar file containing the file web log

### Parameters

- **pprfile** -- Name of the pipeline processing request to be exported. Defaults to a file matching the template 'PPR\_\*.xml'.

Example: `pprfile=['PPR_GRB021004.xml']`

- **targetimages** -- List of science target images to be exported. Defaults to all science target images recorded in the pipeline context.

Example: `targetimages=['NGC3256.band3', 'NGC3256.band6'], targetimages=['r_aqr.CM02.spw5.line0.XXYY.sd.im', 'r_aqr.CM02.spw5.XXYY.sd.cont.im']`

- **products\_dir** -- Name of the data products subdirectory. Defaults to '.'.

Example: `products_dir='..../products'`

### Returns

The results object for the pipeline task is returned.

## Examples

1. Export the pipeline results for a single session to the data products directory

```
>>> !mkdir ..../products
>>> hsdn_exportdata (products_dir='..../products')
```

## 6.2 pipeline.hsdn.cli.hsdn\_importdata

**hsdn\_importdata**(vis: list[str] | None = None, session: list[str] | None = None, hm\_rasterscan: str | None = None, datacolumns: dict | None = None, overwrite: bool | None = None, nocopy: bool | None = None, createmms: str | None = None) → ResultsList[NROIImportDataResults]

Imports Nobeyama data into the single dish pipeline.

Imports Nobeyama data into the single dish pipeline. The hsdn\_importdata task loads the specified visibility data into the pipeline context unpacking and / or converting it as necessary.

If the **overwrite** input parameter is set to False, then when the output MS already exists in the output directory, the existing MS will be imported instead.

### Parameters

- **vis** -- List of visibility data files. These may be MSes, or tar files of MSes.

Example: `vis=['X227.ms', 'anyms.tar.gz']`

- **session** -- List of sessions to which the visibility files belong. Defaults to a single session containing all the visibility files, otherwise a session must be assigned to each vis file.

Example: `session=['Session_1', 'Sessions_2']`

- **hm\_rasterscan** -- Heuristics method for raster scan analysis. Two analysis modes, time-domain analysis ('time') and direction analysis ('direction'), are available.

Default: `None` (equivalent to '`time`')

- **datacolumns** -- Dictionary defining the data types of existing columns. The format is:

`{'data': 'data type 1'}`

or

`{'data': 'data type 1', 'corrected': 'data type 2'}`

For MSes one can define two different data types for the DATA and CORRECTED\_DATA columns and they can be any of the known data types (RAW, REGCAL\_CONTLINE\_ALL, REGCAL\_CONTLINE\_SCIENCE, SELFCAL\_CONTLINE\_SCIENCE, REGCAL\_LINE\_SCIENCE, SELFCAL\_LINE\_SCIENCE, BASE-LINED, ATMCORR). The intent selection strings \_ALL or \_SCIENCE can be skipped. In that case the task determines this automatically by inspecting the existing intents in the dataset. Usually, a single datacolumns dictionary is used for all datasets. If necessary, one can define a list of dictionaries, one for each MS, with different setups per MS. If no type is specified, `{'data': 'raw'}` will be assumed.

- **overwrite** -- Overwrite existing files on import. If `overwrite=False` and the MS already exists in output directory, then this existing MS dataset will be used instead.
- **nocopy** -- Disable copying of MS to working directory.
- **createmms** -- Create an MMS

### Returns

The results object for the pipeline task is returned.

## Examples

- Load MS list in the ./rawdata subdirectory into the context:

```
>>> hsdn_importdata (vis=['../rawdata/mg2-1.ms', '../rawdata/mg2-2.ms'])
```

- Load an MS in the current directory into the context:

```
>>> hsdn_importdata (vis=['mg2.ms'])
```

- Load a tarred MS in ./rawdata into the context:

```
>>> hsdn_importdata (vis=['../rawdata/mg2.tar.gz'])
```

- Import a list of MeasurementSets:

```
>>> myvislist = ['mg2-1.ms', 'mg2-2.ms']
>>> hsdn_importdata(vis=myvislist)
```

## 6.3 pipeline.hsdn.cli.hsdn\_restoredata

**hsdn\_restoredata**(vis: *List[str] = None*, caltable: *VisDependentProperty = None*, refile: *VisDependentProperty = None*, products\_dir: *str = None*, copytoraw: *VisDependentProperty = None*, rawdata\_dir: *str = None*, hm\_rasterscan: *str | None = None*) → Results

Restore flagged and calibration single dish data from a pipeline run.

The hsdn\_restoredata task restores flagged and calibrated data from archived MeasurementSets (MSes) and pipeline flagging and calibration data products.

hsdn\_restoredata assumes that the MSes to be restored are present in the directory specified by the **rawdata\_dir** (default: './rawdata').

By default (**copytoraw** = True), hsdn\_restoredata assumes that for each MS in the input list, the corresponding pipeline flagging and calibration data products (in the format produced by the hsdn\_exportdata task) are present in the directory specified by **products\_dir** (default: './products'). At the start of the task, these products are copied from the **products\_dir** to the **rawdata\_dir**.

If **copytoraw** = False, hsdn\_restoredata assumes that these products are to be found in **rawdata\_dir** along with the MSes.

The expected flagging and calibration products (for each MS) include:

- a compressed tar file of the final flagversions file, e.g. uid\_\_A002\_X30a93d\_X43e.ms.flagversions.tar.gz
- a text file containing the applycal instructions, e.g. uid\_\_A002\_X30a93d\_X43e.ms.calapply.txt
- a compressed tar file containing the caltables for the parent session, e.g. uid\_\_A001\_X74\_X29.session\_3.caltables.tar.gz

hsdn\_restoredata performs the following operations:

- imports the MS(s)
- removes the default MS.flagversions directory created by the filler
- restores the final MS.flagversions directory stored by the pipeline
- restores the final set of pipeline flags to the MS

- restores the final calibration state of the MS
- restores the final calibration tables for each MS
- applies the calibration tables to each MS

When importing the MS, if the output MS already exists in the output directory, the existing MS will be imported instead.

### Parameters

- **vis** -- List of raw visibility data files to be restored. Assumed to be in the directory specified by rawdata\_dir.

Example: `vis=['mg2.ms']`

- **caltable** -- Name of output gain calibration tables.

Example: `caltable='ngc5921.gcal'`

- **reffile** -- Path to a file containing scaling factors between beams. The format is equals to jyperk.csv with five fields:

- MS name
- beam name (instead of antenna name)
- spectral window id
- polarization string
- the scaling factor

Example for the file is as follows:

```
#MS,Beam,Spwid,Polarization,Factor
mg2-20181016165248-181017.ms,NRO-BEAM0,0,I,1.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM0,1,I,1.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM0,2,I,1.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM0,3,I,1.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM1,0,I,3.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM1,1,I,3.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM1,2,I,3.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM1,3,I,3.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM2,0,I,0.5000000000
mg2-20181016165248-181017.ms,NRO-BEAM2,1,I,0.5000000000
mg2-20181016165248-181017.ms,NRO-BEAM2,2,I,0.5000000000
mg2-20181016165248-181017.ms,NRO-BEAM2,3,I,0.5000000000
mg2-20181016165248-181017.ms,NRO-BEAM3,0,I,2.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM3,1,I,2.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM3,2,I,2.0000000000
mg2-20181016165248-181017.ms,NRO-BEAM3,3,I,2.0000000000
```

If no file name is specified or specified file doesn't exist, all the factors are set to 1.0.

Example: `reffile='', reffile='nroscalefile.csv'`

- **products\_dir** -- Name of the data products directory.

Example: `products_dir='myproductspath'`

Default: `None` (equivalent to `'../products'`)

- **copytoraw** -- Copy calibration and flagging tables to raw data directory.

Example: `copytoraw=False`

Default: `None` (equivalent to `True`)

- `rawdata_dir` -- Name of the raw data directory.

Example: `rawdata_dir='myrawdatapath'`

Default: `None` (equivalent to `'./rawdata'`)

- `hm_rasterscan` -- Heuristics method for raster scan analysis. Two analysis modes, time-domain analysis ('time') and direction analysis ('direction'), are available.

Default: `None` (equivalent to `'time'`)

#### Returns

The results object for the pipeline task is returned.

### Examples

1. Restore the pipeline results for a single MS in a single session

```
>>> hsdn_restoredata (vis=['mg2-20181016165248-190320.ms'], reffile='nroscalefactor.csv')
```

## PYTHON MODULE INDEX

### p

`pipeline.h.cli`, 2  
`pipeline.hif.cli`, 5  
`pipeline.hifa.cli`, 29  
`pipeline.hifv.cli`, 72  
`pipeline.hsd.cli`, 94  
`pipeline.hsdn.cli`, 116

# INDEX

## H

h\_init() (*in module pipeline.h.cli*), 2  
h\_resume() (*in module pipeline.h.cli*), 3  
h\_save() (*in module pipeline.h.cli*), 3  
h\_tsyscal() (*in module pipeline.h.cli*), 3  
h\_weblog() (*in module pipeline.h.cli*), 4  
hif\_analyzealpha() (*in module pipeline.hif.cli*), 5  
hif\_applycal() (*in module pipeline.hif.cli*), 6  
hif\_checkproductsize() (*in module pipeline.hif.cli*), 7  
hif\_correctedampflag() (*in module pipeline.hif.cli*), 8  
hif\_editimlist() (*in module pipeline.hif.cli*), 9  
hif\_findcont() (*in module pipeline.hif.cli*), 11  
hif\_lowgainflag() (*in module pipeline.hif.cli*), 12  
hif\_makecutoutimages() (*in module pipeline.hif.cli*), 13  
hif\_makeimages() (*in module pipeline.hif.cli*), 13  
hif\_makeimlist() (*in module pipeline.hif.cli*), 15  
hif\_makersimages() (*in module pipeline.hif.cli*), 18  
hif\_mstransform() (*in module pipeline.hif.cli*), 18  
hif\_rawflagchans() (*in module pipeline.hif.cli*), 19  
hif\_refant() (*in module pipeline.hif.cli*), 21  
hif\_selfcal() (*in module pipeline.hif.cli*), 22  
hif\_setjy() (*in module pipeline.hif.cli*), 24  
hif\_setmodels() (*in module pipeline.hif.cli*), 25  
hif\_transformimagedata() (*in module pipeline.hif.cli*), 26  
hif\_uvcontsub() (*in module pipeline.hif.cli*), 27  
hifa\_antpos() (*in module pipeline.hifa.cli*), 30  
hifa\_bandpass() (*in module pipeline.hifa.cli*), 31  
hifa\_bandpassflag() (*in module pipeline.hifa.cli*), 34  
hifa\_bpsolint() (*in module pipeline.hifa.cli*), 37  
hifa\_diffgaincal() (*in module pipeline.hifa.cli*), 39  
hifa\_exportdata() (*in module pipeline.hifa.cli*), 41  
hifa\_flagdata() (*in module pipeline.hifa.cli*), 42  
hifa\_flagtargets() (*in module pipeline.hifa.cli*), 44  
hifa\_fluxcalflag() (*in module pipeline.hifa.cli*), 45  
hifa\_gaincalsnr() (*in module pipeline.hifa.cli*), 45  
hifa\_gfluxscale() (*in module pipeline.hifa.cli*), 47  
hifa\_gfluxscaleflag() (*in module pipeline.hifa.cli*), 49  
hifa\_imageprecheck() (*in module pipeline.hifa.cli*), 50  
hifa\_importdata() (*in module pipeline.hifa.cli*), 51  
hifa\_lock\_refant() (*in module pipeline.hifa.cli*), 53  
hifa\_polcal() (*in module pipeline.hifa.cli*), 53  
hifa\_polcalflag() (*in module pipeline.hifa.cli*), 54

hifa\_renorm() (*in module pipeline.hifa.cli*), 54  
hifa\_restoredata() (*in module pipeline.hifa.cli*), 56  
hifa\_session\_refant() (*in module pipeline.hifa.cli*), 57  
hifa\_spwphaseup() (*in module pipeline.hifa.cli*), 58  
hifa\_targetflag() (*in module pipeline.hifa.cli*), 61  
hifa\_timegaincal() (*in module pipeline.hifa.cli*), 62  
hifa\_tsysflag() (*in module pipeline.hifa.cli*), 64  
hifa\_tsysflagcontamination() (*in module pipeline.hifa.cli*), 66  
hifa\_unlock\_refant() (*in module pipeline.hifa.cli*), 67  
hifa\_wvrgcal() (*in module pipeline.hifa.cli*), 67  
hifa\_wvrgcalflag() (*in module pipeline.hifa.cli*), 69  
hifv\_analyzestokescubes() (*in module pipeline.hifv.cli*), 73  
hifv\_applycals() (*in module pipeline.hifv.cli*), 73  
hifv\_checkflag() (*in module pipeline.hifv.cli*), 74  
hifv\_circfeedpolcal() (*in module pipeline.hifv.cli*), 75  
hifv\_exportdata() (*in module pipeline.hifv.cli*), 76  
hifv\_exportvllassdata() (*in module pipeline.hifv.cli*), 77  
hifv\_finalcals() (*in module pipeline.hifv.cli*), 77  
hifv\_fixpointing() (*in module pipeline.hifv.cli*), 78  
hifv\_flagcal() (*in module pipeline.hifv.cli*), 78  
hifv\_flagdata() (*in module pipeline.hifv.cli*), 79  
hifv\_flagtargetsdata() (*in module pipeline.hifv.cli*), 80  
hifv\_fluxboot() (*in module pipeline.hifv.cli*), 80  
hifv\_hanning() (*in module pipeline.hifv.cli*), 81  
hifv\_importdata() (*in module pipeline.hifv.cli*), 82  
hifv\_mstransform() (*in module pipeline.hifv.cli*), 83  
hifv\_pbcor() (*in module pipeline.hifv.cli*), 85  
hifv\_plotsummary() (*in module pipeline.hifv.cli*), 85  
hifv\_priorcals() (*in module pipeline.hifv.cli*), 85  
hifv\_restoredata() (*in module pipeline.hifv.cli*), 86  
hifv\_restorepims() (*in module pipeline.hifv.cli*), 88  
hifv\_selfcal() (*in module pipeline.hifv.cli*), 88  
hifv\_semiFinalBPdcals() (*in module pipeline.hifv.cli*), 89  
hifv\_solist() (*in module pipeline.hifv.cli*), 89  
hifv\_statwt() (*in module pipeline.hifv.cli*), 90  
hifv\_syspower() (*in module pipeline.hifv.cli*), 90  
hifv\_testBPdcals() (*in module pipeline.hifv.cli*), 91  
hifv\_vlasetjy() (*in module pipeline.hifv.cli*), 92  
hifv\_vlassmasking() (*in module pipeline.hifv.cli*), 92  
hsd\_applycal() (*in module pipeline.hsd.cli*), 94  
hsd\_atmcor() (*in module pipeline.hsd.cli*), 95

hsd\_baseline() (*in module pipeline.hsd.cli*), 97  
hsd\_blfld() (*in module pipeline.hsd.cli*), 101  
hsd\_exportdata() (*in module pipeline.hsd.cli*), 103  
hsd\_flagdata() (*in module pipeline.hsd.cli*), 103  
hsd\_imaging() (*in module pipeline.hsd.cli*), 105  
hsd\_importdata() (*in module pipeline.hsd.cli*), 106  
hsd\_k2jycal() (*in module pipeline.hsd.cli*), 108  
hsd\_restoredata() (*in module pipeline.hsd.cli*), 109  
hsd\_skycal() (*in module pipeline.hsd.cli*), 111  
hsd\_tsysflag() (*in module pipeline.hsd.cli*), 113  
hsdn\_exportdata() (*in module pipeline.hsdn.cli*), 116  
hsdn\_importdata() (*in module pipeline.hsdn.cli*), 117  
hsdn\_restoredata() (*in module pipeline.hsdn.cli*), 118

## M

module  
  pipeline.h.cli, 2  
  pipeline.hif.cli, 5  
  pipeline.hifa.cli, 29  
  pipeline.hifv.cli, 72  
  pipeline.hsd.cli, 94  
  pipeline.hsdn.cli, 116

## P

pipeline.h.cli  
  module, 2  
pipeline.hif.cli  
  module, 5  
pipeline.hifa.cli  
  module, 29  
pipeline.hifv.cli  
  module, 72  
pipeline.hsd.cli  
  module, 94  
pipeline.hsdn.cli  
  module, 116