HIFI Pipeline Specification

Document: ICC2008-154

Do Kester

Odile Coeur-Joly

Andrea Lorenzani

Martin Melchior

...more...

HIFI Pipeline Specification: Document: ICC2008-154

Do Kester
Odile Coeur-Joly
Andrea Lorenzani
Martin Melchior
...more...

Revision History		
Revision 0.1	1 February 2008	
Outline on Dokuwikki		
Revision 0.6	30 July 2008	
Minor Updates		
Revision 0.7	13 Nov 2009	
Updates to all sections		
Revision 0.8	24 Nov 2009	
Updates to HRS pipeline flow diagram, clarification of MkWbsZero, change to manual properties for new documentation framework		
Revision 0.9	28 July 2010	
Updates to Level-1 and -2 pipeline flow diagram, split "Generic Pipeline" chapter into Level-1 and Level-2 Pipeline chapters, rearrangement and additions of new sections to reflect status of pipeline.		

Table of Contents

Chapter 1. Introduction

This document will be helpful for those wishing to learn about the HIFI pipeline in depth in order to, for example, modify the pipeline algorithms - or to otherwise understand precisely your data has gone through. For those investigating the pipeline for the first time it is recommended to look in the pipeline chapter in the HIFI User Manual first.

Each step of the HIFI data processing pipeline is described here in detail. For each pipeline task the name, purpose, assumptions, mathematics and algorithm is given, the result of each step in terms of changes to HIFI data and metadata is also given, as are possible errors and warnings that can arise.

Chapter 2. Level 0 Pipeline

2.1. DoHkCheck

- **Purpose:**

  The Task checks the house-keeping values defined and contained inside an HifiCalibrationDataset (as `FpuTrendTable`), the result of the check is inserted as flag in the `HifiTimelineProduct` (HTP).

- **Description**

  The Task checks if the House Keeping (HK) values contained in a `HifiCalibrationDataset` (`FpuTrendTable`) are inside the thresholds limits that are defined by the Calibration Scientist and contained in the CalHkThresholds. The result of that checks are inserted as bit flags in the `Column` "hk flag" in the `HifiTimelineProuct`.

  Moreover if some HK value is Out of limit a `BooleanParameter` (= false) with the same name will be added to the `MetaData` of the `HifiSpectrumDataset` and the related quality flag is raised.

  Not all `HifiSpectrumDataset` in the `HifiTimelineProuct` are checked, the default are: {"stab", "hc" , "hot", "cold", "science", "other"}; but this set can be changed with the input parameter "type".

  Several utility methods have been added to the Task to have more user friendly check to be used in the Interactive Analysis of the `FpuTrendTable`.

  A requirement for the Task is that each HTP contains the checks for the Horizontal and Vertical polarization, this implies that HK values raise a different bit in the "hk flag" and that both `CalHkThresholds` for "H" and "V" polarization have to be passed to the Task.

  - Assumptions
    Several assumption are defined for the name convention and the `CalHkThreshods` that will be used from the Task.

    In this document, by "HK mnemonics" we mean the mnemonics that can be used in a `TmSourcePacket` to retrieve the value of a specific House keeping parameter , e.g: HF_AH1_MXBIAS_C.

    By "ColumnName" we mean the name (`String`) of a normal `Column` that is in the `TableDataset`. In the `GenericPipelineCalTable` of the `CalHkThreshods` the `Column` are used to contains the thresholds of some physical quantity. e.g. "mixerCurrentMax" will contain the upper limit of the mixer current in function of the LO frequencies.

    By "ColumnRadix" we mean a sub-String that is in common in the initial part of 2 or more ColumnNames and that refer to the same physical parameter to be checked, e.g. "mixerCurrent" is the ColumnRadix of "mixerCurrent", "mixerCurrentMax", "mixerCurrentMin", "mixerCurrentVar".

    - The difference between a ColumnRadix and a ColumnName is a "suffix" that determine the type of threshold defined in the Column, the suffix recognized from the Task are: "Max" for the upper limit, "Min" for the lower limit, "Var" for the upper limit of the Variance. A ColumnName equal to the ColumnRadix indicate the nominal (expected) measurement for the specific physical quantity. The Variance upper limit can be also defined as a coefficient that multiply the nominal value, in this case that coeffients will be contained in a MetaData DoubleParameter with name: ColumnRadix+"VarCoeff"

- More than one HK mnemonics can be associated to the same ColumnRadix, e.g. "HF_AH1_MXBIAS_C", "HF_AV1_MXBIAS_C"are associated to the ColumnRadix "mixerCurrent".

- The HK mnemonics are associated to a specific bit to be raised in case that it's value is out of limit, however because more than one HK mnemonics can be associated to a physical quantity, (e.g the resistance need the measurement of voltage and current from HK mnemonics) several HK mnemonics can be associated to the same bit.

- The HK mnemonics that refer to the Horizontal polarization start with: "HF_AH" , the HK mnemonics that refer to the Vertical polarization start with: "HF_AV.

- Mathematics

  Simple Mathematics is used from the Task. Some utility method are provided to the user to allow a fast analysis of the Hk mnemonics stored in the `FpuTrendTable`.

  The method `getMean, getMax, getMin` and `getVariance` can return a `Double1d` if the observing time and the integration time are passed as inputs. These will allow to the user to easily compute those values for an HK parameter for all the spectra contained in a `HifiSpectrumDataset`.

- Algorithm

  To work the Task needs 4 Products:

  1) the `HifiTimelineProduct` ("htp") that contains the interval times (the spectra integration time) where the HK parameters have to be monitored.

  2) The `HifiCalibrationDataset` ("cal") that contains all the HK parameters values for the specific observation (e.g the `FpuTrendTable`).

  3-4) The `CalHkThresholds` ("thresholdH" , "thresholdV") that contain the limits for the HK parameters. If these are not passed some default CalHkThresholds are created on the fly. If only one is present it will be used for both polarization.

  The Task checks all the `HifiSpectrumDataset` with the "type" (as retrieved from the `SummaryTable`) that matches one of the `String[]` passed to the Task through the input parameter "type".

  In function of the first observing time value of the `HifiSpectrumDataset` the appropriate `GenericPipelineCalTable`s of the `CalHkThresholds` are retrieved. Note that for each time more than one `GenericPipelineCalTable` type (default is 2) are contained in the `CalHkThresholds`.

  All the ColumnRadix contained in the `CalHkThresholds` are used to check if the related HK parameters are inside the thresholds , i.e. if the values of the parameter during the integration of each spectrum is between the upper and lower limit.

  If the variance suffix "Var" is present in the ColumnName or the variance coefficent is present in the Meta Data, also the Variance of the parameter is checked. This check is made for each single spectrum integration like the upper/lower check, but also the variance of the parameter along all the `HifiSpectrumDataset` observation time is checked.

  While a generic HK mnemonics can be added from the user into the `CalHkThresholds` and it's values will be automatically checked for the upper/lower/variance limits inside the HTP, special case are implemented in the Task for the HK parameters that need some operations before the check is executed. The Column Radix of these parameters are: "magnetResistance","fpuChopper","diplexerResistance".

Of course only the HK mnemonics related to the same polarization are combined to get the final values to check.

- "magnetResistance": The Task assume that there is also a ColumnRadix "magnetCurrent" where it can retrieve the HK mnemonics asscciated to the Magnet Current. The HK mnemonics associated to the "magnetResistance" are related to the Magnet Voltage. The value checked from the Task is the MagnetVoltage/MagnetCurrent.

- fpuChopper": The "chopper check" is to control that the executed chopper position is close to the commanded one. Thus the upper and lower limits of the executed chopper are function of the commanded. The Task checks that the difference between the measured and the commanded chopper positions are inside the thresholds defined. The Task assume that the first HK mnemonics is related to the measured mnemonic and the second is related to the commanded mnemonic.

- "diplexerResistance": The Task assume that the related HK mnemonics that end with "V" have to be used to retrieve the DiplexerVoltage, while the other mnemonics will be used to retrieve the DiplexerCurrent. . The value checked from the Task is the DiplexerVoltage/DiplexerCurrent.

- Calibration Inputs

  The Calibration Inputs of the Task are the `CalHkThresholds`for the Horizontal and Vertical polarizations. and the `FpuTrendTable` containing the HK parameters to be analyzed:

  - **CalHkThresholds**

    The `CalHkThresholds` is a `GenericPipelineCalProduct` thus it has all the basically functionality of that Product. The main difference with a standard `GenericPipelineCalProduct` is that more than one "type" of `GenericPipelineCalTable` can be associated with a specific time. This because the `GenericPipelineCalTable` must have all the columns with the same length, and the HKParameter limits are in function of frequency, thus the parameters can have different number of limits defined/computed in function of frequency. Thus all the HK parameter that have the same number of limits are grouped together in a Table with a specific "type" name. The user can switch between the different tables (i.e. the table returned as default from the `GenericPipelineCalProduct`) through the method `setWorkingType(String tableName)`.

    The `CalHkThresholds` fulfill the same "Assumptions" described in the `DoHkCheck` section .

    The HK monitored are reported in the `MetaData`as `LongParameter`s, with "name" equal the HK mnemonics, the "value" of the parameter is the bit flag associated, and the "description" of the parameter is the ColumnRadix.

    Several methods in the Product help the user to recognize all the relations between the HK mnemonics, the ColumnRadix and the related bit flag.:

    - getWorkingType() the "type" of the `GenericPipelineCalTable` that will be returned as default from the `CalHkThresholds`.

    - getAllTypes() return all the types of `GenericPipelineCalTable` contained.

    - getFlagMap() return the Map that associate each HK Mnemonics to a specific bit flag position.

    - getHkMonitored() return the Map that associate each ColumnRadix to a set (`String1d`)of HK Mnemonics.

- getMnemonic(int bit) return the HK mnemonic associated with a specific bit flag position. If more than one mnemonic have been forced to be associated with the same flag, only the first mnemonic name is returned.
- getMnemonics() return all the HK mnemonic monitored. (in all the tables contained)
- addHkMonitored(String hkMnemonics, String columnRadix, int bit, boolean force) allow to the users to add a new HKmnemonics to be monitored, with the associated columnRadix and the bit flag to use, if the bit is already used a new one is assigned, but in case the method parameter "force"=true.
- getFreeBit() return a bit position that is actually not used from any HK mnemonics.

- **FpuTrendTable**

  The `FpuTrendTable` is an `HifiCalibrationDataset` that contains the values of some HK parameters in function of time for a specific observation. It can be founded in the TrendAnalysis branch of the `ObservationContext` inside the `FpuTrendProduct`.

  The `Column`s of this table have as name the HK mnemonics.

  Specific of this table is the possiblity to set the HK mnemonics to be monitored (thus the Column that will be present in the Table) through the method `setMnemonics(String1d mnemonics)`.

  The pipeline set for default the mnemonics to be monitored with the values retrieved from the `CalHkThresholds.getMnemonics() where the CalHkThresholds is the "H" polarization retrieved from the calibration-tree.`

- **Result:**

  The Task set the new column "hk flag" with the result of all the HK parameters checks performed. If an HK parameters is out of limit a BooleanParameter with the HK mnemonics name is added to the MetaData.

  - MetaData

    All The HK mnemonics can be added as BooleanParameter in the MetaData of `HifiSpectrumDataset` if the relative check on the upper/lower limits is not passed. Also the related ColumnRadix is added as BooleanParameter = false in the MetaDatas of `HifiSpectrumDataset` and in the MetaData of `HifiTimelineProduct`.

    All The HK mnemonics where the variance is checked can be added as BooleanParameter in the MetaData of `HifiSpectrumDataset` with name: "*HKmnemonics* variance"

    In case that the "diplexerResistance" is out of limit only the related "currentDiplexer" mnemonics is set in the metaData (not also the "voltageDiplexer" mnemonics)

  - Columns

    The Task add a new column to the HifiSpectrumDataset where the checks are performed, the Column name is "hk flag" and it will contains the results of all the HK parameters checks performed. The result is a bit raised if the check is not passed. The correspondence between bit and HK Mnemonics can be update/changed from the user changing the `CalHkThresholds`. Below are listed the default correspondences bit position--> HK Mnemonics --> Column Radix

Bit	**HK Mnemonics**	**Column Radix**
0	HF_AH1_MXBIAS_C	mixerCurrent
1	HF_AV1_MXBIAS_C	mixerCurrent

Bit	HK Mnemonics	Column Radix
2	HF_AH1_MXBIAS_V	mixerVoltage
3	HF_AV1_MXBIAS_V	mixerVoltage
4	HF_AH1_MXMG_C	mixerMagnetCurrent
5	HF_AV1_MXMG_C	mixerMagnetCurrent
6	HF_AH1_MXMG_V	magnetResistance
7	HF_AV1_MXMG_V	magnetResistance
8	HF_APR_CH_ROT	fpuChopper
9	HF_DPR_CH_ROT2	fpuChopper
10	HF_AH1_DPACT_V	diplexerResistance
10	HF_AH1_DPACT_C	diplexerResistance
11	HF_AV1_DPACT_V	diplexerResistance
11	HF_AV1_DPACT_C	diplexerResistance
12	HF_AH2_G_FIF1_V	LnaFIF1
13	HF_AV2_G_FIF1_V	LnaFIF1
14	HF_AH2_G_FIF2_V	LnaFIF2
15	HF_AV2_G_FIF2_V	LnaFIF2
16	HF_AH2_G_SIF1_V	LnaSIF1
17	HF_AV2_G_SIF1_V	LnaSIF1
18	HF_AH2_G_SIF2_V	LnaSIF2
19	HF_AV2_G_SIF2_V	LnaSIF2
20	HF_AH2_G_SIF3_V	LnaSIF3
21	HF_AV2_G_SIF3_V	LnaSIF3
22	HF_AP_SCHS_CT	hotLoad
23	HF_AR_SCHS_CT	hotLoad
24	HF_APR_SCCS_CT	coldLoad
25	HF_APR_S2K_CT	level0Temp

- Flags

  The flags raised from this Task are:

  The flags described in the MetaData section above.

  The flags described in the Column section above.

  The following quality flags can be raised in the quality Context in function of the ColumnRadix of the HK mnemonic checked:

  - FPU_MIXER_CURRENT("mixerCurrent","FPU Check: Mixer current is Out Of Limit", false)
  - FPU_MIXER_CURRENT_VARIANCE("mixerCurrentVariance","FPU Check: Mixer current variance is Out Of Limit", false)
  - FPU_MIXER_VOLTAGE("mixerVoltage","FPU Check: Mixer Voltage is Out Of Limit", false)

- FPU_MIXER_MAGNET_CURRENT("mixerMagnetCurrent","FPU Check: Mixer Magnet Current is Out Of Limit", false)
- FPU_MIXER_MAGNET_RESISTANCE("magnetResistance","FPU Check: Mixer Magnet Resistance is Out Of Limit", false)
- FPU_CHOPPER("fpuChopper","FPU Check: chopper measured values differ from the commanded", false)
- FPU_DIPLEXER_RESISTANCE("diplexerResistance","FPU Check: Diplexer Resistance is Out Of Limit", false)
- FPU_LNA("lna","FPU Check: IF Amplifier values are Out Of Limit", false)
- FPU_HOT_LOAD("hotLoad","FPU Check: Hot load temperature is Out Of Limit", false)
- FPU_COLD_LOAD("coldLoad","FPU Check: Cold load temperature is Out Of Limit", false)
- FPU_LEVEL_TEMP("l0Temp","FPU Check: Level 0 Temperature is Out Of Limit", false)

- Calibration Outputs

  None

- Errors and Warnings

  If the input `FpuTrendTable` is missing a SEVERE message is raised: "Trend table is null. No check is possible" ;

  If both inputs for the `CalHkThresholds` are missing a WARNING message is raised: " CalHkThresholds is null. Generated one with default values for both polarization") ;

  If some parameter required from the CalHkThresholds is not present in the `FpuTrendTable` a SEVERE message is raised: "Following paremeters found in Table are not measured (NaN) :"+HKmnemonics

2.2. DoPointingTask

- **Purpose:**

  This module will add the pointing information to each spectrum of the HifiTimelineProduct.

- **Description:**

  The pointing information of the telescope contained in the AuxiliaryContext (part of the ObservationContext) are collected every 0.25s. These data are used together with the observing time and the integration time to associate to of each spectrum the related point information (RA, Dec, position Angle, Velocity). *The parts of this chapter in italics describe functionality is available only from version HCSS 2.0:*

  - Assumptions:
    - The Task requires that the PointingProduct, The OrbitEphemerid and the SiamProduct must be contained inside the AuxiliaryContext passed as input to the Task. *Each one of the Product above can be passed separately to substitute the corresponding one in the AuxiliaryContext. The AuxiliaryContext itself can be skipped if the other three inputs are filled.*
    - Only the HifiSpectrumDataset marked as "isLine" in the SummaryTable will be used to compute the final average values in the MetaData of the HifiTimelineProduct.

- The observationTime and the integration time used are contained in "obs time" (long) and "integration time" (double) columns. The integration time can be a 1 dimensional array (case WBS) or a 2 dimensional array (case HRS), in such a case there is an integration time for each subband. However these times are similar enough that they make no difference for the purpose of the computation of the pointing positions. Thus in case of 2 dimensional array the integration time used is the same for all the subbands, and it is obtained from the average of the times of all subbands. If there are no units in the time columns, it is assumed to be microsecs in the case of "obs time" and seconds in case of "integration time".

- The velocity of the spacecraft reported are calculated in the reference system of LSR. *From version HCSS2.0 it is possible to the user to choose the reference system that will be used, through the input parameter "correction", that will accept the values defined in RadialVelocityTask.Correction , i.e. NONE: the velocity will be given in the reference system of the Earth, EARTHTOSUN will add the correction of the Earth's velocity with respect to the Sun, and SUNTOLSR will add the correction of the velocity of the Sun with respect to the LSR, SUNANDEARTH will add both correction SUNTOLSR and EARTHTOSUN.*

- If the pointing data are obtained without using the integration time the pointing values are obtained with the method getFiltered() from the PointingItem. While in case they are obtained taking in account the integration time they are calculated with the method getGyroPropagated(). *From version HCSS2.0 both case will follow the user choice defined through the input parameter "useGyro".*

- Mathematics:

  If the integration time is used (through the input parameter "useIntegration"): The Right Ascension (R.A.) , Declination (decl) and Position Angle (P.A.) are computed as the weighted average of all pointing information retrieved during the integration. The weight is the inverse of the square of the error.

  The R.A., decl and P.A. reported in the MetaData of each HifiSpectrumDataset is the weighted average of the respective values of the spectrum contained. The weight is the integration time.

  The R.A., decl and P.A. reported in the MetaData of the HifiTimelineProduct is the weighted average of the respective values of the MetaData of the HifiSpectrumDataset where the "isLine" is true. The weight is the total integration time of the HifiSpectrumDataset.

  The errors of RA, dec, Pos Angle reported in the columns before version HCSS2.0 are wrong. *From version HCSS2.0 it is possible to the user to choose the error propagation. As default the error is: the variance of the measurement during the integration time, summed quadratically to the maximum between the errors associated with the pointing measures.*

- Algorithm:

  The PointingProduct, OrbitEphemerisProduct, SiamProduct are retrieved from the AuxiliaryContext passed in the input "aux", *(or from the separated inputs "pointing" ,"orbit" ,"siam": if these are present their value take precedence over the objects found in the AuxiliaryContext.)*

  A loop is performed on all the HifiSpectrumDataset in the HifiTimelineProduct. In each HifiSpectrumDataset the information in the MetaData on "CoordinateSystem" and "equinox" are updated according to the values in the OrbitEphemerisProduct. The R.A., decl and P.A. reported in the MetaData of each HifiSpectrumDataset is the weighted average of the respective values of the spectrum contained. The R.A., decl and P.A. reported in the MetaData of the HifiTimelineProduct is the weighted average of the respective values of the MetaData of the HifiSpectrumDataset where the "isLine" is true.

  For each row of each HifiSpectrumDataset the follow values are calculated and reported in the columns: Right Ascension in "longitude"; Declination in "latitude"; Position Error in "posAngle"; velocity in "velocity"; errors in Right Ascension in "longitudeError"; errors in Declination in

"latitudeError"; and, errors in PosAngle in "PosAngleError". In the case that the PointingProduct contains the information on the raster column number, raster line number, scan line number, and nodding cycle Nnumber, they are added as column information as "rasterColumnNum", "rasterLineNum", "scanLineNum", "nodCycleNum", respectively.

The R.A. Decl, P.A. are calculated in a different way as a function of the input parameter "useIntegration".

- If "useIntegration" is false, the startDate and the endDate are retrieved from the MetaData of the HifiSpectrumDataset. Then an iterator on the PointingProduct with these two dates is created. For each row of the HifiSpectrumDataset the iterator is then used to find the two pointing item closest in time to the observing time. A PointingLinearInterpolator is used to compute the PointingItem at the observation time. From the PointingProduct also the "aperture" at the observation time is retrieved and used together with the Siam to compute the pointed position of each observation.

  This pointed position is obtained in the following way: the PointingItem is obtained with the method getFiltered() (*or getGyroPropagated() if the user sets the parameter "useGyro" = true*) , gives the Quaternion of the uncorrected point. This Quaternion is rotated along the three axes (X,Y,Z) following the values contained in the negation of the Matrix obtained from the Siam , then the Quaternion is transformed in an equivalent Attitude and used to retrieve the information of R.A, decl, P.A.

- If "useIntegration" is true, all the Points measured during the integration time are retrieved and used to obtain a pointed position in the same way as described above (for the "useIntegration"=false). Then all the R.A., dec, and P.A. obtained from these points are averaged weighted by the error associated with each point.

From version HCSS 2.0, the error of R.A., dec, P.A are computed according the user's choice through the parameter "errorType". The default (VARIANCE, or 1) uses the variance of all the point used to calculate a single value, summed quadrically to the maximum error associate to the points used. Other possible option are: (WEIGHTED, or 0) where the final error is calculated taking in account that the inverse of the square of the error is used as a weight to compute the final value of the position. Thus the final error is the inverse of the square root of the sum of the weight. In the option (RANGE, or 2) the associated error is half of the maximum variation of the values, e.g. R.A.error = (MAX(R.A)-MIN(R.A))/2

Note: at the moment the correction in R.A., dec, P.A. , velocities for the chopper angle is NOT included. *(Work in progress to include them in the version HCSS 2.0)*

The OrbitEphemerisProduct, the PointingProduct and the time of each pointing are used to retrieve the projected velocity of the telescope. All the velocity retrieved in the integration time are averaged without any weight. An user set of ephemerids can be passed to the Task through the input parameter "eph".

The velocities are given by default in the LSR reference system. *From version HCSS2.0, the reference system will be calculated in function of the input parameter "correction", that accept the values defined in RadialVelocityTask.Correction*

The MetaData "ra","dec","posAngle" of each HifiSpectrumDataset are filled with the weighted average of the values computed as described above. The weight used is the integration time.

The HifiTimelineProduct MetaData "ra","dec","posAngle" are computed as the weighted average of the MetaData of the HifiSpectrumDatasets that have the attribute "inLine" = true, The weight used is the sum of all the integration times contained in the HifiSpectrumDataset.

- Calibration Inputs:

The Object used as inputs for the Task is the AuxiliaryContext. From it, the task retrieves the PointingProduct, the OrbitEphemerisProduct, and the SiamProduct. (*From version HCSS2.0 that objects can be passed also as stand-alone inputs*).

As optional input, an Ephemerides containing the ephemerides for planets and spacecraft can be passed to the Task. It will be used to compute the velocities of the spacecraft along the observation.

- **Results:**
  - MetaData:

    In the HifiTimelineProduct and in each HifiSpectrumDataset contained the following MetaData are updated:

    - "ra" containing the weighted average of the Right Ascension of the spectra included. For the HifiTimelineProduct only the HifiSpectrumDataset where "inLine" = true are used.

    - "dec" containing the weighted average of the declination of the spectra included. For the HifiTimelineProduct only the HifiSpectrumDataset where "inLine" = true are used.

    - "posAngle" containing the weighted average of the Position Angle of the spectra included. For the HifiTimelineProduct only the HifiSpectrumDataset where "inLine" = true are used.

    - "CoordinateSystem" Containing the coordinate system of the pointing information.

    - "equinox" Containing the equinox used in the pointing information.

    "noPointingInserted" in case there is a failure in the Task to associate the pointing information to the spectra.

  - Columns:

    The following column are added/updated from the Task

    - "longitude" containing the Right Ascension of each spectrum calculated with the algorithm described above;

    - "latitude" containing the Declination of each spectrum calculated with the algorithm described above;

    - "posAngle" containing the Position Angle of the telescope for each spectrum calculated with the algorithm described above;

    - "velocity" containing the velocity of the telescope in the LSR reference system *(in HCSS2.0, in the reference system defined by the input parameter "correction")* for each spectrum, calculated with the algorithm described above;

    - "longitudeError" containing the error of the Right Ascension of each spectrum, calculated with the algorithm described above;

    - "latitudeError" containing the error of the Declination of each spectrum, calculated with the algorithm described above;

    - "PosAngleError" containing the error of the Position Angle of the telescope for each spectrum, calculated with the algorithm described above;;

    - "rasterColumnNum" the column position of the specific spectrum in a raster map ;

    - "rasterLineNum" the line position of the specific spectrum in a raster map ;

- "scanLineNum" the scan line number for the specific spectrum in the on the fly mapping;

- "nodCycleNum" the nodding cycle number for the specific spectrum;

- Flags:

  In case of no pointing information the Task raises in the quality context the flag:"POINTFAIL"

- Calibration Outputs:

  None.

- Errors and Warnings:
  The following SEVERE errors can be raised:

  - In the case of missing OrbitEphemerisProduct: AttitudeReconstructionCategory.NotEnoughData: "OrbitEphemerisProduct cannot be retrieved from AuxiliaryContext and 'orbit' input is null";

  - In the case of missing PointingProduct: AttitudeReconstructionCategory.NotEnoughData: "PointingProduct cannot be retrieved from AuxiliaryContext and 'pointing' input is null";

  - In the case of missing SiamProduct: AttitudeReconstructionCategory.NotEnoughData: " SiamProduct cannot be retrieved from AuxiliaryContext and 'orbit' input is null";

  - In the case of error in the computation of the velocities: AttitudeReconstructionCategory.NotEnoughData" "Problem to retrieve velocity: probably in OrbitEphemerisProduct data are missing.";

  - In the case of a generic exception/error in the Task: AttitudeReconstructionCategory.NotEnoughData "Exceptions while retrieving information from PointingProduct. Latest error was: "+latestPointingException;
  The following WARNING can be raised:

  - In the case there is a failure to retrieve the raster/on the fly position of the spectrum: AttitudeReconstructionCategory.NotEnoughData:"Exceptions while retrieving information from PointingProduct on commanding. " + "Latest error was: "+latestPointingExceptionMeta;

Chapter 3. HRS Pipeline

3.1. Introduction to HRS pipeline

This chapter describes the processing steps involved for data taken specifically with the HRS spectrometer. The steps can be graphically represented in the following figure.

3.2. doHrsSubbands

- Name:

  doHrsSubbands

- Purpose:

  Task that splits the HifiHrsDataFrames into HRS subbands.

- Description:

  The HRS readouts, i.e. the columns "CF" of the HifiTimelineProduct (HTP) are split into HRS subbands.

  At the end of this Task, the Columns "subbandxx" are filled with the raw Correlation Functions of the HRS.

  The channels of the HRS readouts are re-organised, depending on the HRS resolution mode.

  - References:

    CESR-HRS-SP-3162-045.

  - Assumptions:

    The HTP has been already split into several HRS SpectrumDatasets (HSD), according to BBIDs.

    The Columns "configuration" and "blockSelection" are available for all HSDs.

  - Mathematics:

    None.

  - Algorithm:

    Takes in account the 16 HRS configuration words and the Block Selection parameter to find the subbands.

    The HRS configurations indicate how the circuits are chained, how many subbands and channels per subband are available, and for every subband found, which sampler is used in order to find the LO settings for that subband.

  - Calibration Inputs:

    None.

- Result:

  - MetaData:

If subbands have been found, the boolean MetaData "Valid" is added to the HSD and set to True.

The MetaData "subbandstart_xx" and "subbandlength_xx" are re-calculated according to the HRS subbands found.

- Columns:

  For every HSD:

  - as many Columns "subbandxx" are added as subbands found, with xx=1 to 16.

  - the Columns "CF" and "configuration" are removed.

  - the Column "type" (0..4) is added, and corresponds to the mode (correlation, internal test) of the subbands.

  - the Column "sampler" (0..7) is added, and corresponds to the sampler input ASIC of the subbands.

  - the Column "channels" (255...4080) is added, and corresponds to the number of channels of the subbands.

  - the Column "resolution" (0..5) is added, and corresponds to the resolution (ultra-wide-band, wide-band,...) of the subbands.

  - the Column "colorIndex" (0..7) is added, and corresponds to the default Color value when plotting the subbands.

  - the Column "offset" and "duration" are re-ordered according to the subbands order.

- Flags:

  None.

- Calibration Outputs:

  None.

- Errors and Warnings:

  For some HRS functional tests, this step is not allowed (configurations are set to 0).

  If no subband is found, the HRS pipeline can continue after this step but data are not valid.

  In this case, a SEVERE message is sent and the boolean MetaData "Valid" is added to the HSD and set to False.

3.3. doHrsOffsetPow

- Name:

  doHrsOffsetPow

- Purpose:

  Task that computes the Offset and Power of the analog input signal seen by the digital part of HRS.

- Description:

Gets the first channel (channel 0), the integration duration (duration) and the offset of the correlation function, and computes the Offset (mSigma) and Power (vSigma) for this subband.

The column "subbandxx" remain constant after this Task, with raw Correlation Functions of HRS in them.

- References:

  CESR-HRS-SP-3F12-144.

  CESR-HRS-SP-3F12-046.

- Assumptions:

  The step "doHrsSubbands" has been performed.

- Mathematics:

  The estimate of the Power uses the inverse of the error function (erfInv).

- Algorithm:

  digitalOffset = ((offset * 2.0) - duration) / duration

  c0 = channel0 / duration

  t3 = erfInv(2 + digitalOffset - (2.0 * c0))

  t4 = erfInv(2 - digitalOffset - (2.0 * c0))

  vSigma = 0.5 * Math.sqrt(2.0) * t3 + 0.5 * Math.sqrt(2.0) * t4

  mSigma = 0.5 * Math.sqrt(2.0) * t3 - 0.5 * Math.sqrt(2.0) * t4

- Calibration Inputs:

  None.

- Result:

  - MetaData:

    None.

  - Columns:

    For every HSD:

    - the Columns "vSigma" and "mSigma" are added.

    - the Column "offset" is removed.

  - Flags:

    None.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

None.

3.4. doHrsNorm

- Name:

  doHrsNorm

- Purpose:

  Task that normalises the Correlation Functions of HRS.

- Description:

  Correction of the skew introduced by the internal multiplication table of HRS and Normalisation by the first channel (channel 0) of the Correlation Function.

  The Columns "subbandxx" still contain Correlation Functions at the end of this task.

  - References:

    CESR-HRS-SP-3162-045.

    CESR-HRS-SP-3F12-146.

  - Assumptions:

    The steps "doHrsSubbands" and "doHrsOffsetPow" have been performed.

  - Mathematics:

    None.

  - Algorithm:

    Gets the first channel (channel0) of the Correlation Function, and the integration duration (duration).

    c0 = (channel0 * 2.0 - duration) / duration

    normalized_channel = ((raw_channel * 2.0 - duration) / duration) / c0)

  - Calibration Inputs:

    None.

- Result:

  - MetaData:

    None.

  - Columns:

    The Columns "subbandxx" are updated with the normalized Correlation Functions and their description has changed from "rawCF" to "normCF".

    The Column "integration time" is added and filled with the integration duration converted into seconds, according to the formula:

integration_time = duration * Math.pow(2, 9) / (LO7 * 1.e6) (where LO7 is the frequency value of the Local Oscillator of HRS)

- Flags:

  None.

- Calibration Outputs:

  None.

- Errors and Warnings:

  None.

3.5. doHrsQDCFull

- Name:

  doHrsQDCFull

- Purpose:

  Task that corrects the quantization distortion of the correlation functions of HRS.

- Description:

  Gets the Correlation Function (CF) for each subband. Gets the CalHrsQDCFull Product, and then interpolates on the 3 dimensional table of this Product the CF for each correlation channel.

  The Columns "subbandxx" still contains Correlation Functions at the end of this task.

  - References:

    CESR-HRS-SP-3F12-143.

  - Assumptions:

    The step "doHrsNorm" was performed before.

  - Mathematics:

    Analog offset is obtained after 3D interpolation of a value read into the correction table.

  - Algorithm:

    1) Computes vector for the 3D interpolation.

    Takes as input one value of the Correlation Function and the corresponding thresholds mSigma and vSigma.

    Provides the corrected values of : the Correlation Function, mSigma and VSigma.

    These corrected values are the 3D input vector (x,y,z) before interpolation with the correction table.

    2) Computes the Tri-linear interpolation of one input value(x,y,z), according to a 3D grid.

  - Calibration Inputs:

CalHrsQDCFull.

- Result:
  - MetaData:

    None.
  - Columns:

    For every HSD:

    - the description of the Columns "subbandxx" has changed from "normCF" to "corrCF".
  - Flags:

    NOQDC.
  - Calibration Outputs:

    None.
  - Errors and Warnings:

    The CalHrsQDCFull calibration product is mandatory as input.

    If CalHrsQDCFull is not present or its contents is invalid, the HRS pipeline can continue after this step but:

    - HRS data can not be calibrated,
    - no correction is applied,
    - the quality flag NOQDC is raised,
    - a SEVERE message is displayed into the console.

3.6. doHrsPowCorr

- Name:

  doHrsPowCorr
- Purpose:

  Task that applies the gain non-linearity correction to the power of HRS.

  The Columns "subbandxx" still contain Correlation Functions at the end of this task.
- Description:

  Gets the input signal power, gets the CalHrsPowCorr Product, and then corrects the input signal power with this Product.
  - References:

    CESR-HRS-SP-3F12-316.
  - Assumptions:

    The step "doHrsQDCFull" was performed before.

- Mathematics:

  The corrected value of the power is obtained after 1D interpolation of a value read into the correction table.

- Algorithm:

  1) Reads the gain correction into the correction table.

  2) Computes linear interpolation of one input value(x), according to a 1D grid.

- Calibration Inputs:

  CalHrsPowCorr.

- Result:

  - MetaData:

    None.

  - Columns:

    The Column "vSigma" is removed at the end of this Task, and is replaced by "corrVSigma".

  - Flags:

    NOPOWCOR.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    The CalHrsPowCorr calibration product is mandatory as input.

    If CalHrsPowCorr is not found or its contents is invalid, the HRS pipeline can continue after this step but:

    - HRS data can not be calibrated,
    - no correction is applied,
    - the quality flag NOPOWCOR is raised,
    - a SEVERE message is displayed into the console.

3.7. doHrsWindow

- Name:

  doHrsWindow

- Purpose:

  Task that applies a Hanning windowing on the Correlation Functions of the HRS.

  The Task parameter "window" is **optional** and set to **"none"** by default.

- Description:

For each subband, gets the Correlation Function (CF) and applies a Hanning windowing of on all channels of the CF of the HRS.

The Columns "subbandxx" still contain Correlation Functions at the end of this task.

The equivalent Task for applying a Hanning windowing on HRS spectra is named "doHrsSmooth".

- References:

  None.

- Assumptions:

  The step "doHrsPowCorr" was performed before.

- Mathematics:

  Hanning algorithm.

- Algorithm:

  length = size of the Correlation Function

  hann = 0.5 * (1.0 + Math.cos(Math.PI * i / length))

  Each channel of the Correlation Function is multiplied by hann

- Calibration Inputs:

  None.

- Result:

  - MetaData:

    None.

  - Columns:

    The Columns "subbandxx" are updated with the CF after hanning windowing, and their description has changed from "corrCF" to "winCF".

  - Flags:

    None.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    None.

3.8. doHrsSymm

- Name:

  doHrsSymm

- Purpose:

Task that symmetrises the Correlation Function of HRS and add zeros to it if necessary.

The Task parameter "zeros" is **optional** and set to **"multiple"** by default.

- Description:

  For each subband, gets the Correlation Function (cf), duplicates the channels of the CF except channel0, inverts them, and adds them to the CF.

  A modifier allows the user to choose if the set of zeros added to the CF is a "multiple" of 2 or a "power" of 2.

  At the end of this Task the Columns "subbandxx" still contains Correlation Functions

  - References:

    None.

  - Assumptions:

    The step "doHrsPowCorr" was performed before.

    The step "doHrsWindow" is not mandatory before this Task.

  - Mathematics:

    None.

  - Algorithm:

    Duplicates the channels of the CF except channel0, inverts them, and adds them at the end of the CF.

  - Calibration Inputs:

    None.

- Result:

  - MetaData:

    None.

  - Columns:

    The Columns "symmCF" are added.

    The Columns "subbandxx" remained unchanged

  - Flags:

    None.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    None.

3.9. doHrsFFT

- Name:

  doHrsFFT

- Purpose:

  Task that applies a FFT processing on the Correlation Function to in order to obtain the HRS spectra.

  The Task parameter "algo" is **optional** and set to **"FFT"** by default.

  The Task parameter "algo" is **obsolete**, as the best FFT algorithm is automatically selected according to the array size.

- Description:

  For each subband, gets the Column "symmCF", gets the type of FFT to apply and applies it to the contents of symmCF.

  At the end of this Task, the Columns "flux_xx" are created and contain raw HRS spectra.

  - References:

    None.

  - Assumptions:

    The step "doHrsSymm" was performed before.

  - Mathematics:

    The FFT algorithm used is found in ia.numeric.toolbox.

  - Algorithm:

    1) Convert data into Complex1d

    Complex1d fftData = new Complex1d(data)

    2) Compute FFT in place

    FFT.fft(fftData)

    3) Take only the real part of the FFT result

    spectrum = fftData.getReal().get(new Range(0, data.getSize() / 2))

  - Calibration Inputs:

    None.

- Result:

  - MetaData:

    None.

  - Columns:

    The Columns "symmCF" are removed.

The Columns "subbandxx" are removed

The Columns "flux_xx" are added.

- Flags:

  None.

- Calibration Outputs:

  None.

- Errors and Warnings:

  None.

3.10. doHrsSmooth

- Name:

  doHrsSmooth

- Purpose:

  Task that applies a Hanning smoothing on the spectra (equivalent to Hanning on Correlation Function).

  The Task parameter "window" is **optional** and set to **"hanning"** by default.

  No other type of windowing is implemented yet.

- Description:

  For each subband, gets the HRS spectra and applies a Hanning windowing on all spectral channels.

  The equivalent Task for applying a Hanning windowing on HRS Correlation Function is named "doHrsWindow".

  - References:

    None.

  - Assumptions:

    The step "doHrsFFT" was performed before.

  - Mathematics:

    Hanning smoothing.

  - Algorithm:

    Set the first channel to 0.

    Replace all x channels by : x = 1/4 (x+1) + 1/2 (x) + 1/4 (x-1)

    Keep the last channel unchanged

  - Calibration Inputs:

    None.

- Result:
  - MetaData:

    None.

  - Columns:

    The Columns "flux_xx" are updated with the smoothed spectra, and their description has changed from "rawSP" to "Smoothed Spectra".

  - Flags:

    None.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    None.

3.11. doHrsFreq

- Name:

  doHrsFreq

- Purpose:

  Task that computes the frequency of the HRS spectra.

  The Task parameter "model" permits to choose between the creation of frequency columns or the creation of the HRS frequency Linear Model.

  The Task parameter "model" is **optional** and set to **"column"** by default.

- Description:

  If the model is chosen, the parameters of the model are the LOs values and the sampler name for every spectrum.

  If the columns are chosen, this task will create the frequency columns corresponding to the HRS spectra.

  For USB spectra, the frequency values are increasing, for LSB spectra, frequency values are decreasing.

  The MetaData "channelSpacing" is added for later processing during level2, and computed from the LO7 value of HRS and the size of the spectra.

  The cut limits of each spectrum is processed according to the filter response, and will be used later by the Task DoHrsCutBandEdges.

  - References:

    None.

  - Assumptions:

The step "doHrsFFT" was performed before.

- Mathematics:

  None.

- Algorithm:

  None.

- Calibration Inputs:

  None.

- Result:

  - MetaData:

    The parameter channelSpacing is added, computed from: (LO7 / 2) / number of channels.

    If "model" is used, the MetaData "model" is filled with "HRS: linear 1D".

  - Columns:

    The Columns "frequency_xx" are added, unit = MEGAHERTZ.

  - Flags:

    None.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    None.

3.12. doHrsCorrSP

- Name:

  doHrsCorrSP

- Purpose:

  Task that corrects HRS spectra from IF non-linearity errors.

- Description:

  For each spectrum, gets the number of channels and the corrected power, and applies 2 correcting factors to the spectrum.

  The Task parameter "in_db" is **optional** and set to **0** by default.

  If the Task parameter "in_db" is set to 1, the spectrum is computed relatively to the nominal power in dB. In this case no further pipeline processing is allowed, but this mode is used to display QLA spectra.

  - References:

None.

- Assumptions:

  The steps "doHrsFFT" and "doHrsPowCorr" were performed before.

- Mathematics:

  None.

- Algorithm:

  1) Scaling factor1 : divide all channels by the size of the spectrum, except for channel0 which is divided by (number of channels * 2).

  2) Scaling factor2 : multiply all channels by thresholds value and divide them by power vSigma:

  spectrum = spectrum * Math.pow(THRESHOLDS_VALUE / vSigma, 2), with THRESHOLDS_VALUE = 0.160 V.

- Calibration Inputs:

  None.

- Results:

  - MetaData:

    None.

  - Columns:

    The Columns "flux_xx" are updated with the corrected spectra, and their description has changed to "Corrected Spectra".

  - Flags:

    None.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    None.

3.13. doHrsCutBandEdges

- Name:

  doHrsCutBandEdges

- Purpose:

  Task that cuts the edges of the HRS spectra, according to the bandpass of the filters.

- Description:

  Removes the first channel of the spectrum.

Gets as input the "cuts" frequency limits processed by DoHrsFreq, and truncates the spectrum edges.

Updates accordingly the MetaData subbandstart_xx and subbandlength_xx.

Creates the frequency columns even if a HRS frequency model has been used before.

- Assumptions:

  The Task DoHrsFreq has been performed.

- Mathematics:

  None.

- Algorithm:

  None.

- Calibration Inputs:

  None.

- Result:
  - MetaData:

    The MetaData subbandstart_xx and subbandlength_xx are updated after the cut of the spectra.

  - Columns:

    The columns "frequency_xx" and "flux_xx" have a different size after the truncation.

  - Flags:

    None.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    None.

Chapter 4. WBS Pipeline

4.1. Introduction to the WBS pipeline

This chapter describes the processing steps involved for data taken specifically with the WBS spectrometer. The steps can be graphically represented in the following figure.

4.2. DoWbsScanCount

- Name:

  DoWbsScanCount

- Purpose:

  Normalize the integration time of all spectra to 10 milliseconds.

- Description:

  The flux columns of all WbsSpectrumDatasets contained in a HifiTimelineProduct are divided by the scan count column values. The scan count value is the number of integration frames stored in a single ccd readout. Each frame is equivalent to an integration time of 10 milliseconds. Thus the division is equivalent to a normalization of all spectra to an integration time of 10 millisecond.

  A new column named "integration time" is added to the HifiSpectrumDataset. It contains the integration time of each spectrum in seconds. These values are calculated as: the scan count divided the number of scans made in one second, i.e. 100.

  The "dark" column, which contains the dark measurement values, is also divided by the scan count values.

  - Assumptions:

    Each scan count is equivalent to an integration time of 10 milliseconds.

    The second bit of the MetaData "Pipeline applied" flag is applied.

  - Mathematics:

    A simple division of "flux" and "dark" columns by "scancount" is performed.

  - Algorithm:

    The Task works with spectra that are split into subbands and also unsplit spectra.

    There is an internal check to verify if the scan count division has already been applied to the spectra: the MetaData "Pipeline applied" is retrieved, the second bit is checked and if the value is equal to 1 the ScanCount division is not applied. If the MetaData "Pipeline applied" doesn't exist a new one is created.

    After the division is performed the second bit of "Pipeline applied" will be set to 1.

    This implementation will be used until a proper History mechanism can be used. At this moment (January 2008) the history mechanism doesn't work with the HifiTimelineProduct.

  - Calibration Inputs:

No Calibration Input is needed.

- Result:
  - MetaData:

    The MetaData "Pipeline applied" is checked; if it does not exist, it is created.

    The second bit of this MetaData is set to 1 after the Task is applied

  - Columns:

    Columns "flux" (or "flux_subband") and "dark" are changed.

    A new Column, "integration time", is created.

  - Flags:

    No flag is set.

  - Calibration Outputs:

    None.

  - Errors and Warnings:

    If the data are already corrected a warning Quality Message is raised: QCFlags.DataProcessing.UnprocessedData.

    If the scancount value is less or equal to zero a warning Quality Message is raised: QCFlags.DataProcessing.CalibrationIssues.WrongExposureCorrection.

4.3. MkWbsBadPixels

- Name:

  MkWbsBadPixels

- Purpose:

  Checks for saturated pixels. Sets the "saturated" flags in the spectra.

- Description:

  The task goes through the WbsSpectrumDataset and checks the flux values of each pixel. If the flux value is above the physical range of possible values in a 10 bit CCD (i.e. (2^10)-1=1023), the flag of that specific pixel at that specific time is marked as "saturated".

  If a specific pixel is flagged as saturated more than 90% of the time (this default can be modified), then the pixel is flagged as "bad".

  The output of the Task is a calibration product, CalWbsBadPixel. CalWbsBadPixel contains an array in which all "bad" pixels calculated from all WbsSpectrumDatasets in the HifiTimelineProduct are merged in a 1-Dimensional array.

  A previously calculated "bad pixel mask", CalWbsBadPixel, can be also used as input. The output will be an array with the merged "bad" pixels of both masks.

  - Assumptions:

The CalWbsBadPixel used as input (if one is not supplied a default is generated) indicates:

- Threshold value for pixel saturation.
- Threshold value (as a percentage) of number of scans in which a pixel may be marked as saturated before it is marked as "bad".
- The value of the flag for saturation.

- Mathematics:

  The logic operation "OR" is used between flags.

  The Task also defines a logical operation (method public static) makeOr for Int1d and also for an array with different lengths. The "OR" between integer is the standard java "|" operator, where the numbers are converted to their binary representation.

  When merging arrays of different lengths, the unmatched elements of the longest array are copied (unchanged) in the result.

- Algorithm:

  The Task retrieves the "bad pixel Mask", the threshold values and the saturated flag value from CalWbsBadPixel, which is passed in the "cal" input. If the user does not set a specific CalWbsBadPixel, a default one is used.

  For each WbsSpectrumDataset the following operation are performed:

  - Each pixel of each spectrum is checked against the saturation threshold values
  - From the results of the previous operation, the Task sets the "flag" column in the WbsSpectrumDataset: if that column already exists, an "OR" operation is performed between the newly calculated flags and the existing flags; otherwise, a new column is created.
  - The Task adapts the bad pixel mask to the values for the band starts and length of the considered WbsSpectrumDataset
  - Then it calculates the number of times that each pixel is saturated inside each WbsSpectrumDataset.
  - The Task computes an array, "saturated mask", that indicates which pixels have been flagged as saturated in more than 90% (this default can be changed) of spectra.
  - The "saturated mask" is merged with the "bad pixel mask" in a new "bad pixel mask".

  The new "bad pixel masks" computed from all WbsSpectrumDatasets are merged in the resulting "bad pixel mask". This final "bad pixel mask" will be put in the resulting output.

  The Task works with spectra split into subbands and also unsplit and also unsplit spectra. The result is always a single array equivalent to joined ccds

- Calibration Inputs:

  The calibration product, CalWbsBadPixel, which is either one generated by default or a previously calculated "bad pixel mask".

- Results:
  - MetaData:

No new MetaData are defined or changed.

- Columns:

  Sets or creates "flag"/"flag_N" column/s.

- Flags:

  The Saturated flag value is retrieved from CalWbsBadPixel.

  The CalWbsBadPixel saturation value is the static variable SATURATION_VALUE with value =2 (January 2008).

- Calibration Outputs:

  The CalWbsBadPixel passed in the input is merged with the bad pixel mask calculated from saturated pixels.

- Errors and Warnings:

  None

4.4. DoWbsBadPixels

- Name:

  DoWbsBadPixels

- Purpose:

  Masks bad pixels.

- Description:

  Task that applies the masking pixel list. Configurable to select/unselect masks.

  - Assumptions:

    The following definition (from January 2008) of WbsSpectrumDataset subbands is used:

    When the subbands are joined, the resulting array has always a length = 2048*4 =8192. (Numbers defined in the DefaultValues class). When the subbands are split, their lengths are given from the subbandlength_X MetaData, thus the total number of the channels of all subbands can be less than 8192.

    If the task marks the unobserved channels with a flag (value = CalWbsBadPixel.NOT_OBSERVED), when the WbsSpectrumDataset is split into subbands the corresponding flag columns will be also split and the above flag value will be removed. If the WbsSpectrumDataset subbands are split and rejoined, the flag values of the unobserved channels will be lost.

    The first 4 pixels of each CCD, used for the dark calibation, are marked with the appropriate flag: HifiMask.DARK_PIXEL

  - Mathematics:

    The logic operator "OR", as defined in the makeOr method of the Task MkWbsBadPixels, is used.

- Algorithm:

  The Task works with unsplit spectra and also when the spectra are divided into subbands.

  If the WbsSpectrumDataset is not split into subbands, the unobserved channels are marked with the value CalWbsBadPixel.NOT_OBSERVED.

  The MetaData "isMasked" is created with the value defined by the Task input parameter "apply".

  For each row of the flags of all WbsSpectrumDataset contained in the HifiTimelineProduct a logical OR is performed with the flags of the "Bad Pixel Mask" passed in the Input.

- Calibration Inputs:

  The calibration product CalWbsBadPixel containing the "bad pixel map" calculated in MkWbsBadPixels.

- Result:

  - MetaData:

    The MetaData "isMasked" (BooleanParameter) is created, with the value defined by the input parameter "apply".

  - Columns:

    Set or create "flag"/"flag_N" column/s.

  - Flags:

    The Not Observed Flag is set in unobserved channels if the data are not split into subbands, the value of the flag is retrieved from CalWbsBadPixel.NOT_OBSERVED.

    A logical OR is performed with the flag contained in the CalWbsBadPixel passed as Input.

    The first 4 pixels of each CCD are Marked with the appropriate flag: HifiMask.DARK_PIXEL

  - Calibration Outputs:

    None.

  - Errors and Warnings:
    There is notification message when there are unobserved channels.

4.5. DoWbsDark

- Name:

  DoWbsDark

- Purpose:

  Subtract the dark values from the "flux" values.

- Description:

  Each subband has 4 values that contain the dark measured for the specific spectrum. The odd and the even pixels have different dark values. Thus the appropriate darks are selected for the subtraction from the flux values. The user can select which dark has to be used for the subtraction:

"darkKind"	Odd Channels:	Even Channels:
DARK1_2=0	The 1st dark pixel (default).	The 2nd dark Pixel (default).
DARK3_4=1	The 3rd dark pixel.	The 4th dark pixel.
DARK_AVERAGE=2	The average of the 1st and 3rd dark pixels.	The average of the 2nd and 4thdark pixels.

- Assumptions:

  The first four pixels of each subband contain the dark values (if the subbands have not been cut).

  In case of discrepancy between these values and the dark values, the values used are a function of the input parameter "usePixel". If true (default) the values of the pixels are used, but only if they are different from 0, else (usePixel=0) the "dark" values will be used.

- Mathematics:
  The task performs a simple subtraction of the dark values from all channels.

- Algorithm:

  The Task works with both spectra split into subbands and unsplit spectra.

  There is an internal check to verify if the dark subtraction has already been applied to the spectra: the MetaData "Pipeline applied" is retrieved, the third bit is checked and if the value is equal to 1 the dark subtraction is not applied. If the MetaData "Pipeline applied" does not exist a new one is created.

  If the subband has been cut, the first channel in the subband could be an even pixel [not an odd one]. The darks for the even/odd channels are selected as a function of the real channel position inside the ccd.

  The first four pixels of each subband contain the dark values (if the subband has not been cut). In case of discrepancy between these values and the dark values, the values used are a function of the input parameter "usePixel". If true (default) the values of the pixels are used, but only if they are different from 0, else the "dark" values will be used.

  After the subtraction is performed the third bit of "Pipeline applied" will be set to 1.

  This implementation will be used until a proper History mechanism can be used. At this moment (January 2008) the history mechanism doesn't work with the HifiTimelineProduct.

- Calibration Inputs:

  No Calibration Input passed.

- Result:

  - MetaData:

    The MetaData "Pipeline applied" is checked. If it does not already exist, it is created.

    The third bit of this MetaData is set to 1 after the task is applied

  - Columns:

    Column(s) "flux" (or "flux_X") is (are) changed.

  - Flags:

  - Calibration Outputs:

None.

- Errors and Warnings:

  There is an internal check between the first four pixels of each subband and the dark values in the "dark" column. If they have different values a warning message is raised: QCFlags.DataProcessing.CalibrationIssues.WrongOffseDarkCurrentCorrection.

  If the Task is applied to data already dark-subtracted a warning message is raised advising that the dark correction will be NOT applied.

4.6. DoWbsNonLin

- Name:

  DoWbsNonLin

- Purpose:

  Perform the correction for the non-linearity in response of the CDD for even and odd WBS pixels.

- Description:

  - Assumptions:

    The ccd pixels have a non-linearity that can be corrected with a polynomial.

    There are different corrections for odd and even pixels.

    The measured coeffients of the polynomial needed for the non-linearity correction are stored in a calibration product, CalWbsLinearCoeff

    Before the non-linearity correction, the pixels have been bit-shift corrected, scan-count corrected and dark-subtracted.

  - Mathematics:

    The pixel count values are divided by their maximum possible value (1023)

    Then two polynomials are applied to the even and odd pixels. The new flux (f') of each channel will be a function of the old flux (f) and the coeffients of the polynomial(C_0,C_1, C_2, C_3), i.e.:

    $f'=C_0+C_1*f+C_2*f^2+C_3*f^3$

    Finally the pixel count values are multiplied by their maximum possible value (1023).

  - Algorithm:

    The Task works with spectra both split into subbands and also unsplit.

    There is an internal check to verify if the non-linearity correction has already been applied to the spectra: the MetaData "Pipeline applied" is retrieved, the 4th bit is checked and if its value is already equal to 1 the non-linearity correction is not applied. If the MetaData "Pipeline applied" does not exist, a new one is created.

    If the subband has been cut, the first channel in the subband could be an even channel (not an odd one). The coefficients for the even/odd channels are selected as a function of the real channel position inside the ccd.

    All channels are normalised, i.e they are divided by the maximum count value that a pixel can have(1023). This assumes that the scan count and dark correction has already been applied.

For each subband the relative polynomials are applied to the even and odd pixels.

All channels are then multiplied by the maximum value that a pixel can have(1023).

After the correction is performed the fourth bit of "Pipeline applied" will be set to 1. This implementation will be used until a proper History mechanism can be used. At this moment (January 2008) the history mechanism does not work with the HifiTimelineProduct.

- Calibration Inputs:
  The calibration product CalWbsLinearCoeff contains the coefficients for the polynomial of each subbands.

- Result:
  - MetaData:

    The MetaData "Pipeline applied" is checked. If it does not already exist, it is created.

    The fourth bit of this MetaData is set to 1 after the Task is applied

  - Columns:
    Column(s) "flux" (or "flux_X") is (are) changed. The non linearity correction is applied.

  - Flags:
    No flags changed from this Task.

  - Calibration Outputs:
    None.

  - Errors and Warnings:

    If the Task is applied to data already corrected for the non-linearity, a warning message advises that the correction will be NOT applied.

4.7. MkWbsZero

- Name:

  MkWbsZero

- Purpose:

  Check the zeros and compute an interpolation to represent the zero correction as a function of time.

- Description:
  - Assumptions:

    The first row (of each pair of rows) in the "Combs" dataset is used as the "Zero" if the On Board Software version is greater than or equal to the version 4.03.03. (Condition met during routine operations.)

    The "Zeros" wil be extracted from the "Combs" and used *unless* the On Board Software version is less than version 4.03.03, in which case the Zeros that are present in the HifiTimelineProduct will be used.

    The String parameter allowed for the interpolation are the String fields defined in herschel.ia.toolbox.spectrum.utils.interp.InterpolRule.

    The threshold values for the "Zero" checks are contained in the Quality product QWbsZero.

- Mathematics:

  Each "Zero" spectrum is checked:

  - The value of each channel should be inside the range determined from the minimum(QWbsZero.getThresholdMin()) and maximum(QWbsZero.getThresholdMax()) values allowed.

  - The average of the Zero should be inside the range determined from the average minimum(QWbsZero.getThresholdAverageMin()) and maximum(QWbsZero.getThresholdAverageMax()) values allowed.

  - If there is more than one "zero", all the "Zeros" after the first should pass a variance check: the variance between a Zero and the previous one should be below a threshold value (QWbsZero.getThresholdVariance()). The variance between two Zeros is calculated as:

  # [(Z_2-Z_1)- # Z_2- Z_1#] ^{2}#

  where Z_2 and Z_1 are the Zero "flux", where only the channels with the flag value (in Z_1 or Z_2) different from 0 are used.

  No interpolation is performed in this Task. It just sets the interpolation parameter in the Calibration.

- Algorithm:

  The Task detects the number of the "zero" SpectrumDatasets in the HifiTimelineProduct.

  If there are not any "zero" SpectrumDatasets the Zeros from "comb" are used. If the "comb" datasets are also missing the Task returns an error message and exits.

  A WbsSpectrumDataset containing all the Zeros of the HifiTimelineProduct is assembled.

  The Zeros are checked by the following procedure:

  The threshold ranges are retrieved from the QWbsZero passed in the Input/Output.

  A mask is created for each zero spectrum as a function of the flag pixel value equal to 0.

  The zero "flux" values are retrieved and masked.

  For each zero it calculates the minimum, the maximum and the average.

  If the values calculated are inside the allowed range, the zero is marked as "good"

  If there is more than one zero spectra in the dataset, the variance of the zeros following the first is calculated and checked. The variance is checked in the following way: The previous "good" zero is retreived, the masks of the 2 zeros are merged (with AND operation), and then the two fluxes are masked with this merged mask. Then, the variance between the two zeros is calculated with the formula above. Finally, the task checks that the variance value is below the threshold value.

  Only the zeros that pass also the variance check are marked as "good".

  The Task sets the results in the Quality Product QWbsZero.

The "good" and "bad" zeros (the zeros that didn't pass the checks) are put in the CalWbsZero as two different HifiTimelineProducts.

The task sets the interpolation in the CalWbsZero as a function of the input parameter "interp" of the Task. If the interpolation type is not recognized, an error of type CalibrationIssuesCategory.WrongFlatFielding. is returned.

Finally the Task sets the the CalWbsZero in the "cal" output and the QWbsZero in the "zeroCheck" output.

- Calibration Inputs:
  A QWbsZero is used to retrieve the threshold values. If no QWbsZero is passed through the "zeroCheck" parameter, a default QWbsZero is created.

- Result:
  - MetaData:

    Set the BooleanParameter "checkZero" equal to the same MetaData parameter of QWbsZero.

  - Columns:
    No changes in the HifiTimelineProduct.

  - Flags:
    No changes in the HifiTimelineProduct.

  - Calibration Outputs:

    A new calibration Product, CalWbsZero.

    A Quality Product QWbsZero in the "checkZero" output.

  - Errors and Warnings:

    If there are no "zero" spectra in the HTP an error, CalibrationIssuesCategory.WrongFlatFielding, is raised.

    If the interpolation required is not implemented an error, CalibrationIssuesCategory.WrongFlatFielding, is raised.

4.8. DoWbsZero

- Name:

  DoWbsZero

- Purpose:

  Subtract the zero spectra from all spectra present in the observation. The zero spectra from other observations can also be used, if required.

- Description:

  The zeros from the zero calibration measurement are interpolated and subtracted from the spectra in the observation. The possible interpolations are those defined in InterpolRule.

  - Assumptions:

    A CalWbsZero containing the zero is passed to the Task.

- Mathematics:

  The zeros are interpolated as a function of time using the ArrayInterpolator. A simple subtraction is then performed.

- Algorithm:

  The Task works with both spectra that are split into subbands, and also spectra that are unsplit.

  If the zeros contained in the calibration are split into subbands, they are joined to calculate the appropriate channel position for the interpolation. After the subtraction is performed the bands are split again, if they were previously divided.

  There is an internal check to verify if the zero subtraction has already been applied to the spectra: the MetaData "Pipeline applied" is retrieved, the fifth bit is checked and if its value is equal to 1 the zero subtraction is not applied.

  If the CalWbsZero is null or has empty data the spectra are unchanged and a warning message, DataProcessingCategory.UnprocessedData, is raised.

  For each spectrum of the observation a zero is obtained from an interpolation of the zeros in the calibration: if the interpolation has type NEAREST or PREVIOUS the appropriate Zero is selected, if the interpolation has a different type an ArrayInterpolator is created with the times and fluxes of the calibration, then the ArrayInterpolator is used to calculate the Zeros for all the spectra.

  The obtained zeros are subtracted from the flux of the spectra.

  After the subtraction is performed the fifth bit of "Pipeline applied" will be set to 1. If the MetaData "Pipeline applied" doesn't exist a new one is created.

- Calibration Inputs

  A CalWbsZero obtained from the present observation or from another observation.

- Result:

  - MetaData:

    The MetaData "Pipeline applied" is checked. If it does not yet exist, it is created.

    The fifth bit of this MetaData is set to 1 after the Task is applied.

  - Columns:
    Column(s) "flux" (or "flux_X") is (are) changed. The zero subtraction is applied.

  - Flags:
    No flags changed by this Task.

  - Calibration Outputs:
    None.

  - Errors and Warnings:

    If the Calibration is empty a warning message, DataProcessingCategory.UnprocessedData, advises that the zero correction has not be performed.

    If the Task is applied to data that has already been corrected for the zero subtraction a warning message, DataProcessingCategory.UnprocessedData, advises that the correction will be NOT applied.

4.9. MkWbsFreq

- Name:

  MkWbsFreq

- Purpose:
  Derives the frequency scale from the comb spectra.

- Description:

  This task uses the comb spectra to derive the scale for frequency calibration, and defines how to proceed if there are errors with the comb measurement.

  There is a difference between "comb spectra with some errors" (e.g. some comb line are not detected) and "no fitted comb". The latter will return an exception when the Task attempts to fit them as no information is available from that comb.

  To use "comb spectra with some errors", the Signature parameter "use_bad" must be set as as true.

  HRS data can be also used for frequency calibration of the WBS. At the moment, HRS data will only be used instead of Combs to create the WbsFreqCal if none of the combs can be fitted, or if no combs are present in the observation.

  - Assumptions:

    The first row (of each pair of rows) in the "comb" dataset is a "zero" if the On Board Software version is greater equal to the version: 4.03.03. (This condition is met in routine operations.)

  - Mathematics:

    The Median and the Rms of the CCDs are calculated with the Basic.MEDIAN and Basic.RMS of the numeric package.

    The Task uses the LevenbergMarquardtFitter with a Gaussian Model and the Fitter with a PolynomialModel from herschel.ia.numeric.toolbox.fit to fit the data.

  - Algorithm:

    The Task checks the number of Comb present.

    - If there is at least one fitted comb, the Task writes to the output calibration Product, CalWbsFreq, the computed CalWbsFreqCoeff and the input CalWbsFreqTuning.

    - Otherwise, it will try to use the HRS data to create a CalWbsFreq and to set it as output. See the section below "Get frequency Calibration from HRS Algorithm".

    Creating CalWbsFreq from the comb:

    - The task removes the "zero" row(s) from all combs dataset and creates a new WbsSpectrumDataset containing only the combs [called wbsSpectrum in this document].
    - If the wbsSpectrum still has a "flux" column it will be split into subbands;
    - Then the Task checks each row of the wbsSpectrum to find the approximate position of the lines. See the section below "Rough Check Channel Algorithm".
    - A rough check is then made on channels to find possible spikes and missed lines. See the section below "Spikes and missed line Algorithm".
    - Then each row of the wbsSpectrum is fitted. See the section below "Fitting Comb Algorithm".

- From the fitting result, the task fills the data in the Quality product QWbsFreq.
- The MetaData "checkComb", "spikeNumber", "spikeNumberFlaf" in the HifiTimelineProduct are also set to tag the quality of the Comb contained.
- If the wbsSpectrum was originally unsplit, the CCDs are then rejoined.
- From the fitting result, the Task creates the calibration Product, CalWbsFreqCoeff, to be inserted in the Frequency Calibration.
- The quality product, QWbsFreq, is set in the output "combCheck".

Rough Check Channel Algorithm:

- This is a check on each single subband to find the approximate line positions in each CCD.
- The Task calculates the median of the ccd.
- If the threshold value input is zero the task calculates the threshold value: it calculates the rms of the CCD, then recalculates it after removing pixels with values three times or more greater than the previous rms. The threshold value is the median value plus three times the second rms calculated.
- A number of pixels equal to CalWbsFreqTuning.getStartCcd() and CalWbsFreqTuning.getEndCcd() are removed from the borders of the CCD.
- All masked pixels are removed from the CCD.
- The remaining pixels are scanned to find possible lines.
- A line candidate is found if a channel fulfills all following conditions: The value minus the median is greater than the threshold , the value is greater than the neighbors pixels with distance =1, the values of the pixels at a distance less or equal to 5 minus the square root of the median are smaller of the values of a Gaussian with the peak in the examined pixels and a sigma equal to CalWbsFreqTuning.getLineWidth().
- If the number of lines detected in this way is greater than the number of line expected plus the maximum number of spike allowed the Task returns a warning CalibrationIssuesCategory.WrongWavelengthScaleGain.
- If the number of lines detected is equal to the number of line expected plus one, the first or the last line will be not used. The line not used is the line with a position, respect to the border of the ccd, with a distance smaller than half of the expected distance between 2 lines [i.e. CalWbsFreqTuning.getLineStep()]. A warning message CalibrationIssuesCategory.WrongWavelengthScaleGain is raised.

Spikes and missed line Algorithm:

- The possible line positions founded from the Rough Check Channel Algorithm are checked to found if there are spikes or if some lines are missed.
- The parameters used are: the expected distance between two lines CalWbsFreqTuning.getLineStep() [here:lineStep] and the tollerance allowed in this distance CalWbsFreqTuning.getLinesStepTollerance() [here:lineTol]
- In general to identify spikes and missed lines two consecutive guessed lines are examined. If they are the first two lines of the ccd (ordered from left to right) the next line is taken in account to identify which one of the two lines is the spike or the missed line, else the second line is marked as spike/missed line.

- If the distance between two consecutive guessed lines is greater than (2 x lineStep - linetol), it is identified as missed line. A warning message CalibrationIssuesCategory.WrongWavelengthScaleGain is raised. A guessed line position is added to the possible line position (with position equal to the first line position plus the lineStep value).

- If the distance between two consecutive detected lines is smaller than (lineStep - linetol), the line is identified as spikes. A warning message CalibrationIssuesCategory.WrongWavelengthScaleGain is raised. The spike position is removed from the possible line positions.

- If the distance between two consecutive guessed lines is greater than (lineStep + linetol) and smaller than (2 x lineStep - linetol), they are identified as missed line plus a spikes. A warning message CalibrationIssuesCategory.WrongWavelengthScaleGain is raised. The spike position (the second line as above) is removed from the possible line positions. A guessed line position is added to the possible line positions (with position equal to the first line position plus the value of lineStep).

- If the final number of line detected is smaller than the expected number of line a warning message CalibrationIssuesCategory.WrongWavelengthScaleGain is raised.

Fitting Comb Algorithm:

- For each comb the Task loops on the subbands to fit each CCD.

- It use the possible line positions founded with the rough Check Channel Algorithm and the Spikes and missed line Algorithm.

- It select a region of pixels equal to 2 X CalWbsFreqTuning.getGaussianRange() around each line.

- It remove the masked pixels from these regions.

- It subtract the minimum of the region from the pixel values.

- It fit the resulting regions with a LevenbergMarquardtFitter with a GaussModel where the initial values for each region are: {the maximum value of all selected regions, the position of the maximum of the specific region, the expected line width}.

- From the LevenbergMarquardtFitter it calculates the amplitude of the gaussians, the resolution of the gaussians, the standard deviation of the gaussian fit, the power of the gaussians and the position of the peak of each gaussian.

- Only gaussians with amplitude above the threshold are used in the following steps. This avoids that possible line positions, inserted from the Spikes and missed line Algorithm, are used when the lines are effectively not present.

- It use a PolynomialModel with degree equal to CalWbsFreqTuning.getPolynomialDegree() to fit the line positions in function of the expected line frequency values.

- It use a PolynomialModel with degree equal to CalWbsFreqTuning.getPolynomialDegree() to fit the expected line frequency values in function of the line positions.

- It store in the results the RMS of the frequency fit.

- It obtains the powerReduct removing the first and last line from the line powers.

- It calculates the ripple as: 10*Log(Max(powerReduct)/MIN(powerReduct))

- It obtains the dynamic range calculated as: (2 * scanCount * 1023/ RMS(noise)) with the following meaning of the symbols: *scanCount* is the number of scan Count added in the frame and it is proportional to the total integration time of the frame; RMS(noise) is the RMS calculated from the pixels that are not used to fit the lines.
- The efficiency is calculate as the mean of the powerReduct.
- The resolution is calculated as the mean of the resolution of the lines.

Get frequency Calibration from HRS Algorithm.

TBD: Work in progress

- Calibration Inputs

A CalWbsFreqTuning is used to retrieve all the parameter needed for the fitting of the comb lines and for the threshold parameters used in the Quality Product.

The Signature parameters "COMB_FIRST_LINE_POSITION","COMB_LINES_STEP","COMB_THRESHOLD" are user convenience shortcut to overwrite the equivalent parameters in the input CalWbsFreqTuning.

An Hrs HifiTimelineProduct containing the spectra to be used in the calibration of Wbs with Hrs.

- Result
  - MetaData

    Set the BooleanParameter "checkComb" equal to the same MetaData parameter of QWbsFreq

    Set the LongParameter "spikeNumber" equal to the same MetaData parameter of QWbsSpikes

    Set the BooleanParameter "spikeNumberFlag" equal to the same MetaData parameter of QWbsSpikes

  - Columns

    No changes in the HifiTimelineProduct.

  - Flags

    No changes in the HifiTimelineProduct.

  - Calibration Outputs:

    CalWbsFreq, containing the CalWbsFreqTuning and the CalWbsFreqCoeff, is found in the "cal" output.

    In the "checkComb" output: A quality QWbsFreq that is a MapContext containing a QWbsComb for each comb present in the observation. Each QWbsComb is also a MapContext that contains 4 QWbsCcd with the quality result of the fit of the ccds.

  - Errors and Warnings:

    CalibrationIssuesCategory.WrongWavelengthScaleGain error and/or warning problems are found in the comb.

4.10. DoWbsFreq

- Name:

DoWbsFreq

- Purpose:

  Applies the frequency Calibration to all data in the observation.

- Description:

  The frequency calibration supplied in the input is used to create a Wbs2DFreqModel, then from the model the frequencies are calculated and set as new Columns in the HifiTimelineProduct.

  - Assumptions:

    A CalWbsFreq is passed to the Task.

    The CalWbsFreq contains the CalWbsFreqTuning and the CalWbsFreqCoeff.

    If a null frequency Calibration is passed, the Task will create and apply a default Frequency calibration. The default frequency calibration is created with the frequency coefficients defined inside the defaults of CalWbsFreqCoeff.

  - Mathematics:

    A Wbs2DFreqModel is used to calculate the frequencies.

  - Algorithm:

    The Task works with spectra that are both split into subbands and also still unsplit.

    For the interpolation in time, if the first time in the WbsFreqCal is uequal to zero then all the times of the calibration will be shifted by the time value of the first spectrum in the HifiTimelineProduct.

    The Wbs2DFreqModel is created in function of the number of combs present in the observation, the time parameters of the calibration, the start and the end of each subband and the degree of the polynomial set in the calibration.

    Then the model is set in each WbsSpectrumDataset

    Then the method toTabulated() in the WbsSpectrumDataset is called to create the new Colums.

    Then the model is removed.

    The MetaData "Pipeline applied" in the WbsSpectrumDataset is retrieved, the six bit is set equal to 1.

  - Calibration Inputs

    A CalWbsFreq obtained from the present observation or from another observation.

- Result

  - MetaData

    The sixth bit of the MetaData "Pipeline applied" is set equal to 1 after the Task is applied.

  - Columns

    New columns containing the frequencies are added. The number of culumns added is equal to the number of "flux" Columns. The basic name of the columns is defined by the method WbsSpectrumDataset.toTabulated() that retrieve the information from the Metadata "wave-name".

- Flags

  No flags changed from this Task.

- Calibration Outputs

  None.

- Errors and Warnings

  If the Calibration is empty a warning message CalibrationIssuesCategory.WrongWavelengthScaleGain advice that a standard frequency calibration will be applied

  If the Calibration times start from zero a warning message CalibrationIssuesCategory.WrongWavelengthScaleGain advice that the time of the calibration will be shifted to the present time of the observation.

4.11. MkWbsFluxAtten

- Name:

  MkWbsFluxAtten

- Purpose:

  Provides the technical CalWbsAttSpecific that will be used to calculate the Intensity calibration as a function of the attenuator settings.

- Description:

  A CalWbsAttSpecific is created as a function of the Attenuator measurement in the HTP.

  - Assumptions:

    The CalBbid contains the name "att" to indicate the bbtype of the attenuators.

    The "att" SpectrumDatset will contain Columns with names: "HW_IN_ATT" and "Band_ATT"

    Attenuator spectra are in the sequence ABABAB...

  - Mathematics:

    The resulting spectrum for each pair of spectra, A and B, is calculated as

    (Flux_a_i/Flux_b_i)/(att_i_B-att_i_A), where

    Flux_a_i is the flux of the subband i of the attenuator spectra A,

    Flux_b_i is the flux of the subband i of the attenuator spectra B,

    att_i_A is the attenuation coefficient for subband i of the attenuator spectrum A,

    att_i_B is the attenuation coefficient for subband i of the attenuator spectrum B.

  - Algorithm:

    If no CalWbsAttSpecific is provided a default one is created.

    The Task will loop on all SpectrumDatset contained in the HTP to find the attenuators bbtype.

The Task will use a CalBbid to retrieve the appropriate bbtype in each SpectrumDataset.

The row positions of the attenuator spectra are recorded.

For each pair of attenuator spectra the following procedure is applied:

From the Column "HW_IN_ATT", the Task retrieves the attenuation values for the spectra of type A and B. Then it calculates the general attenuation coefficients as a function of the last 4 bits. If b0,b1,b2,b3 identify the last 4 bits start counting from the right, the result is:

att = b0 + b1*2+b2*4+b3*8.

From the Column "Band_ATT" the Tasks retrieves the attenuation values of the specific subb-bands. Then it calculates the specific band attenuation coefficients as a function of the last 3 bits. If b0,b1,b2 denote the last 3 bits start counting from the right, the result for the band i is:

att_i = att+ b0*1 + b1*2+b2*4.

Where att is the general attenuator coefficient.

The resulting spectrum is calculated as (Flux_a_i/Flux_b_i)/(att_i_B-att_i_A), as above.

In the case that att_i_A is equal to att_i_B a CalibrationIssuesCategory.WrongExposureCorrection is raised.

The time of the resulting spectrum is calculated as the average of the times of the spectra A and B

All the resulting spectra are written in CalWbsAttSpecific in the output.

- Calibration Inputs:

  A CalWbsAttSpecific containing the basic multiplication factors for the operations on the bits.

- Result:
  - MetaData:

    No new MetaData are defined or changed in the HTP.

  - Columns:

    No new Columns are defined or changed in the HTP.

  - Flags:

    No new flags are defined or changed in the HTP.

  - Calibration Outputs:

    A CalWbsAttSpecific containing the spectrum coefficients (one for each channel for each couple of attenuator spectra) to be used in the Intensity calibration.

  - Errors and Warnings:

In the case that the attenuator coefficents for the specific band of the spectra A and B are equal, a CalibrationIssuesCategory.WrongExposureCorrection is raised.

4.12. DoWbsSubbands

- Name:

  DoWbsSubbands

- Purpose:

  Split the spectra into subbands.

- Description:

  Perform splitInCcd() on all WbsSpectrumDataset contained in the HifiTimelineProduct.

  - Assumptions:

    The name of the frequencies column is given by the SpectrumDataset method, getWaveName().

  - Mathematics:

    None.

  - Algorithm:

    A loop on all WbsSpectrumDataset contained in the HifiTimelineProduct is performed.

    If the WbsSpectrumDatasets is not already split, the method spliInCcd is invoked.

    If the WbsSpectrumDatasets have the frequencies column they are split using the method splitIn-Segments.

  - Calibration Inputs:

    None.

- Result:

  - MetaData:

    No MetaData are defined or changed in the HTP.

  - Columns:

    The Column "flux" is replaced by 4 columns with names "flux_1","flux_2","flux_3","flux_4".

    The Column "flag" is replaced by 4 columns with names "flag_1","flag_2","flag_3","flag_4".

    The Column "frequency" (or the name returned from the method getWaveName) is replaced by 4 columns with names "frequency_1","frequency_2","frequency_3","frequency_4".

  - Flags:

    No Flags are changed in the HTP.

  - Calibration Outputs:

    None.

- Errors and Warnings:

  No Errors are thrown by this Task.

4.13. MkSpur

- Name:

  MkSpur

- Purpose:

  Checks for spurs in the cold calibration spectra. Returns a proposed list of spur channel candidates.

- Description:

  The Task is a translation to Java of the jython script WbsSpurFinderTask Written by C. Borys

  This routine will loop through all hot/cold datasets in a HifiTimeLineProduct and catalogue spurs. It is best run after the WBS branch but before the generic pipeline.

  - Assumptions:

    That hot and cold are contained only in "hc" datasets and they have only 2 rows.

    The first row is the cold load, and the second row is the hot.

  - Mathematics:

    See the Algorithm section.

  - Algorithm:

    The Hot-Cold SpectrumDatset (or the passed SpectrumDataset) are retrieved and copied from the HifiTimelineProduct. Then a Smoothing with filter type = "box" and "width"=3 is applied to the spectra.

    First, all fluxes above the saturation threshold are flagged. The routine then determines the width of each saturated region, cataloging them as they are found. A region larger than the saturated pixels is flagged as bad, since the wings of the spur are not saturated yet clearly part of the spur. The excess region flagged is 75% the width of the spur on either side of it.

    Next, the second derivative of the flux is calculated and its RMS determined. Using a threshold of sigma*RMS, the point in the second derivative that deviates the most is found, and a Gaussian is fit to the original flux. The spur is flagged, and then the process repeated for other points in the second derivative until none are found that deviate more than the threshold.

    The spurs are returned as a table dataset, and flag column in each hot/cold dataset in the HTP are set appropriately.

    The option to pass a Spectrum2d to the Task instead of an HifiTimelineProduct has been added as an optional input. The Task will perform the calculations and flag the value for the first row of the passed Spectum2d. This is never used in the pipeline, but it is required to use the task interactively.

  - Calibration Inputs:

    No Calibration Input are passed to the Task.

    However several input parameter can be passed:

- "threshold": The flux above which the WBS is considered saturated
- "sigma": The threshold used to determine if something is a spur is given by sigma*X, where X is the RMS of the second derivative of the flux. The default of 10.0 was determined from TV/TB and Gascell observations.
- "fwhm": Spurs are fit using a Gaussian profile. The initial guess at the width of the spur is given by this variable. The default of 15 was determined from TV/TB and Gascell observations.

- Result:
  - MetaData:
    None.
  - Columns:
    None.
  - Flags:
    The flag bit (with value defined from "HifiMask.SPUR_CANDIDATE") is set in the flags for the channels in which a spur candidate is found.
  - Calibration Outputs:

    The output of the task is a CalSpur Object, which is a Product containing a TableDataset with name "spur", and with the following Columns:
    - "obsid" a Long1d that contains the obsid from the MetaData of the SpectrumDataset analyzed.
    - "hcid" a Long1d that contains the "ds_id" from the MetaData of the SpectrumDataset analyzed.
    - "apid" a Long1d that contains the apid from the MetaData of the SpectrumDataset analyzed.
    - "Band" a String1d that contains the "band" from the MetaData of the SpectrumDataset analyzed.
    - "LO" a Double1d that contains the values of the "LO Frequency" of the first row of the SpectrumDataset analyzed.
    - "subband" a Long1d that contains the subband analyzed
    - "Pixel" a Double1d that contains the pixel centre of the spur obtained from the Gaussian fit
    - "IF" a Double1d that contains the frequency centre of the spur obtained from the Gaussian fit
    - "amp" a Double1d that contains the amplitude of the spur obtained from the Gaussian fit
    - "width" a Double1d that contains the width in MHz of the spur obtained from the Gaussian fit
    - "Type" a String1d that contains the type of the spur (negative or positive) obtained from the sign of the amplitude of the spur
  - Errors and Warnings:
    A NullPointerException is thrown if neither a dataset or a HTP have been passed to the task.

Chapter 5. Generic Pipeline

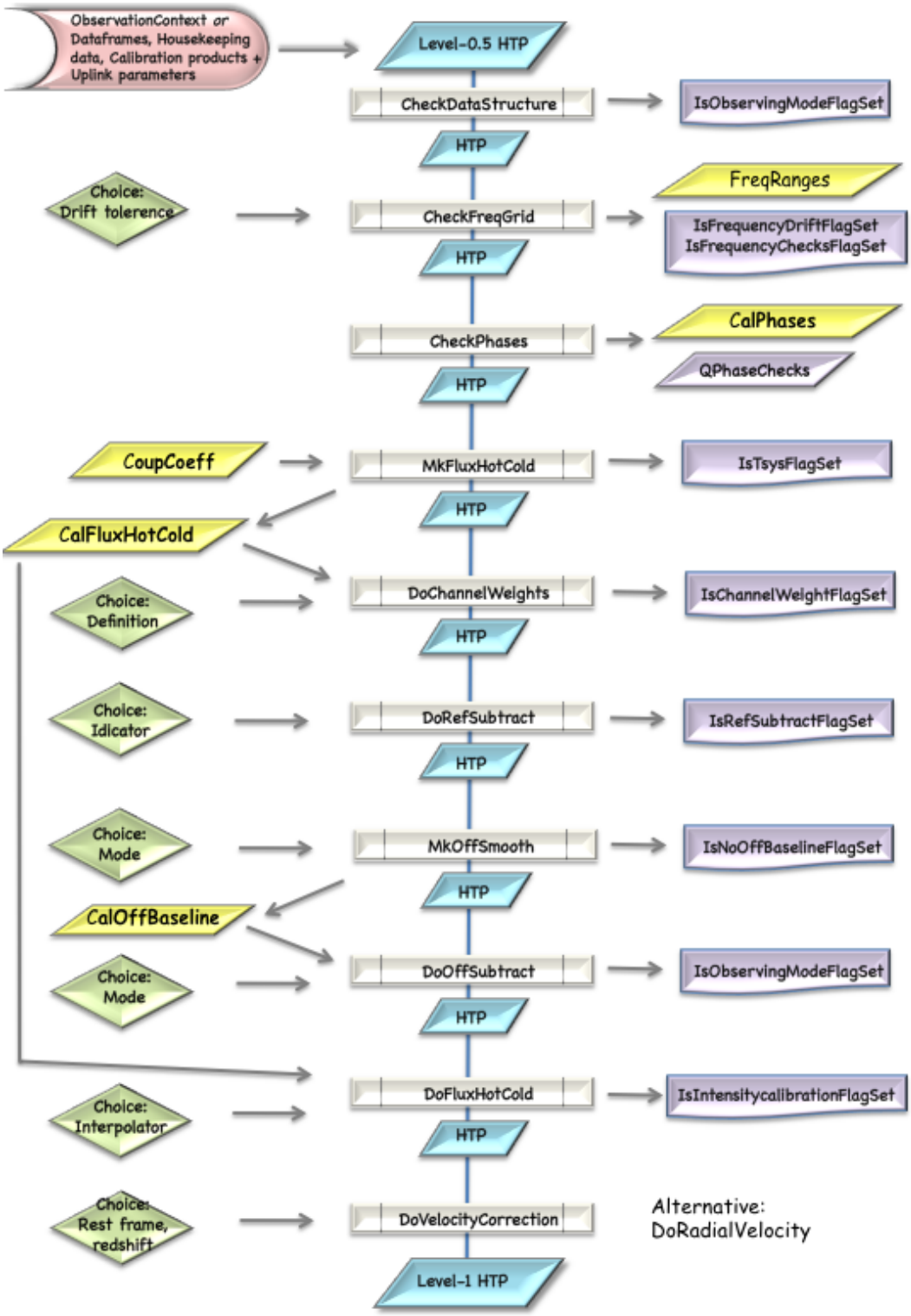
5.1. Preliminaries

We start with some first remarks on what data the pipeline is configured to work with.

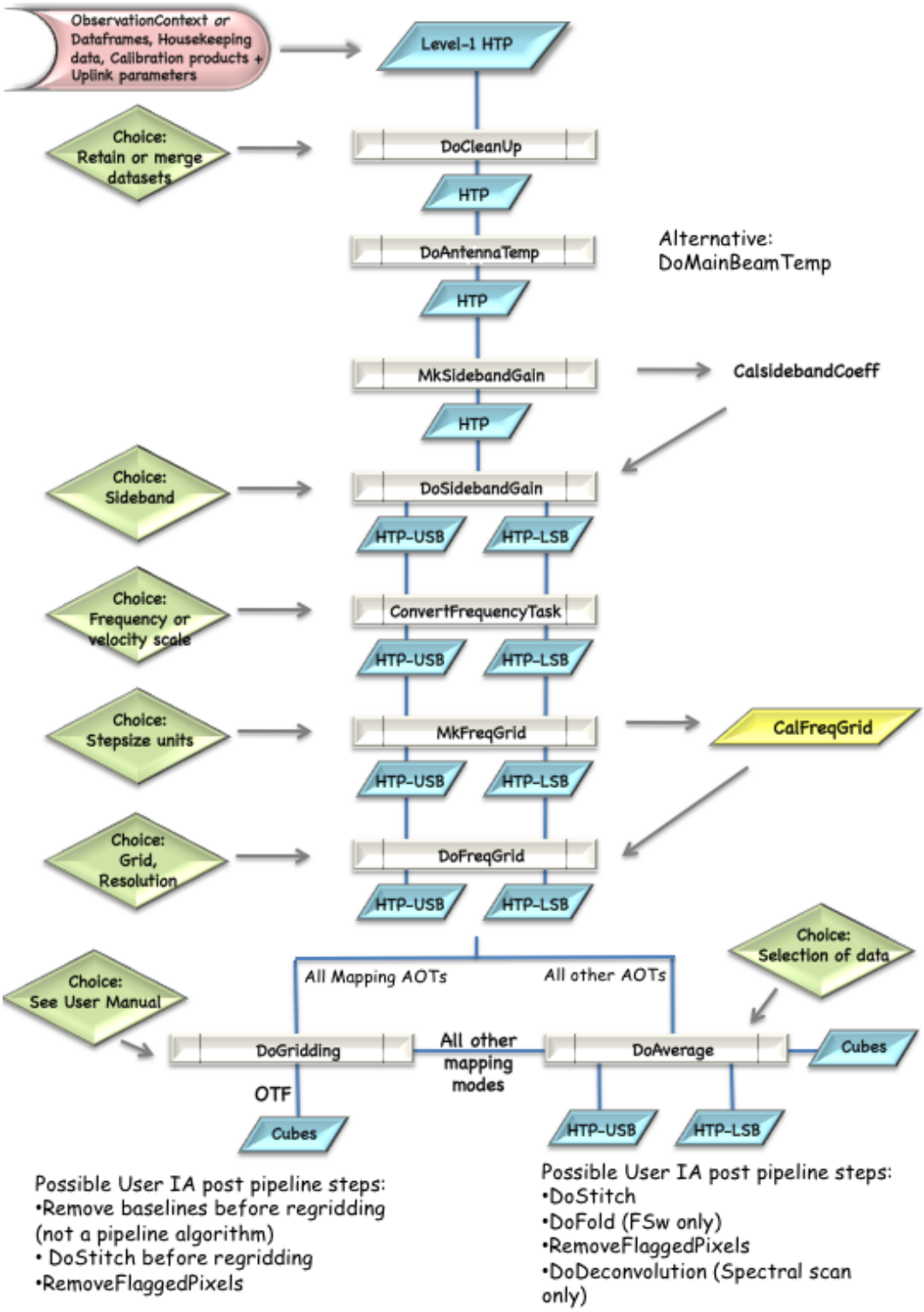
5.1.1. Introduction to the Generic Pipeline

This chapter describes the processing steps involved for data already processed to Level 0.5 to Levels 1.0 and 2.0.

The two figures provide a graphical representation of the steps.



And the final processing steps...



5.1.2. Configuration of the Generic Pipeline

The generic pipeline consists of a sequence of tasks (modules) that are configured in a observing mode specific way. There are two alternatives to specify the pipeline for specific observing modes:

- Distinguish the different handling of the observing modes in the python script by rather lengthy `if...elseif ...else` statements.

- Generate, a `PipelineConfiguration`-object (see `herschel.hifi.pipeline.generic`-package) that contains all the observing mode specific parameters. By passing this object to the tasks, the task-specific parameters can be retrieved from it and the task is processed accordingly. The `PipelineConfiguration`-object can be generated by a statement of the form

```
config = PipelineConfiguration.getConfig(htp)
```

  where `htp` stands for an instance of a `HifiTimelineProduct`.

- A third more user friendly alternative will consist in opening a suitable GUI that allows the user to view the default configuration for a given `HifiTimelineProduct` (once a default is available) and change the settings if desired.

In the following, we concentrate on the second alternative.

When calling `PipelineConfiguration.getConfig(htp)`, by default the parameters specified in the file *PipelineConfig.xml* in `herschel.hifi.pipeline.generic`-package are loaded. As a user you can inspect what defaults are configured by calling from the console `print config` or for a specific task (such as "doRefSubtract") `print config.getParameters("doRefSubtract")`. You can even modify the entries by `config.setParameter("doRefSubtract","indicator","Chopper")` . Note that changes will be effective only for this specific config object loaded by the `getConfig`-method. Expert users can load custom configurations with the default task parameters by setting the system property `hifi.pipeline.generic.config` to the suitable file with the custom configuration. **Warning:** The location of the file should be included in the system variable "CLASSPATH".

In the following, a typical section of such an xml file is shown:

```
 <group name="DBS" modes="HifiPointModeDBS, HifiMappingModeDBSRaster,
HifiMappingModeDBSCross,
HifiSScanModeDBS">
   <task name="doRefSubtract">
     <param name="isABBA" type="Boolean">True</param>
     <param name="startWithRef" type="Boolean">True</param>
            <param name="offStartsWithOpposite" type="Boolean">True</param>
            <param name="ignore" type="Boolean">False</param>
          </task>
          <task name="mkOffSmooth">
            <param name="ignore" type="Boolean">True</param>
          </task>
          <task name="doOffSubtract">
            <param name="ignore" type="Boolean">False</param>
           <param name="mode">addOff</param>
          </task>
 </group>
```

As you see, we rather refer to "groups" of observing modes where in each group the parameters for a group of observing modes may be configured. A group is defined between the tags `<group>...</group>` and by giving it a name (`name="DBS"`) and specifying the `modes` it should be associated with.

Within a group, the parameters for a given task can be specified by first opening a `<task>`-tag and then using the `<param>...</param>`-tags.

5.1.3. Standard Observing Modes

We next give an overview on the observing modes that are configured by the system and hence can **automatically** be processed by running the generic pipeline. The (default) configuration pa-

rameters for these observing modes are specified in the *PipelineConfig.xml* file in the package `herschel.hifi.pipeline.generic`-package.

	Observing Mode	Label	bbtype
I-1	Position switch reference	HifiPointModePositionSwitch	6021 (OFF), 6022(ON)
I-2	Slow chop DBS	HifiPointModeDBS	6031 (ON), 6032(OFF)
I-2a	Fast chop DBS	HifiPointModeFastDBS	6042 (ON), 6043(OFF)
I-3	Frequency switch (with position switch reference: OFF)	HifiPointModeFSwitch	6038 (ON), 6039(OFF)
I-3a	Frequency switch no ref (without position switch reference: no OFF)	HifiPointModeFSwitchNoRef	6038 (ON)
I-4	Load chop (with position switch reference: OFF)	HifiPointModeLoadChop	6035 (ON), 6036 (OFF)
I-4a	Load chop no ref (without position switch reference: no OFF)	HifiPointModeLoadChopNoRef	6035 (ON)
II-1	On the fly map	HifiMappingModeOTF	6021 (ON), 6022 (OFF)
II-2	Slow chop DBS raster map	HifiMappingModeDBSRaster	6031 (ON), 6032 (OFF)
II-2a	Fast-chop DBS raster map	HifiMappingModeFastDBSRaster	6042 (ON), 6043 (OFF)
II-2b	Slow chop DBS cross map	HifiMappingModeDBSCross	6031 (ON), 6032 (OFF)
II-2c	Fast-chop DBS cross map	HifiMappingModeFastDBSCross	6042 (ON), 6043 (OFF)
II-3	Frequency switch OTF map (with position switch reference: OFF)	HifiMappingModeFSwitchOTF	6038 (ON), 6039 (OFF)
II-3a	Frequency switch no ref OTF map (without position switch reference: no OFF)	HifiMappingModeFSwitchOTFNoRef	6038 (ON)
III-2	Slow chop DBS scan	HifiSScanModeDBS	6031 (ON), 6032 (OFF)
III-2a	Fast chop DBS scan	HifiSScanModeFastDBS	6042 (ON), 6043 (OFF)
III-3	Frequency switch scan (with position switch reference: OFF)	HifiSScanModeFSwitch	6038 (ON), 6039 (OFF)
III-3a	Frequency switch no ref scan	HifiSScanModeFSwitchNoRef	6038 (ON)

	(without position switch reference: no OFF)		

5.1.4. Observing Modes Groups

From the viewpoint of the generic pipeline, the following groups of AOT's are identical:

- Position Switch: I-1
- DBS: I-2, II-2, II-2b, III-2
- Fast DBS: I-2a, II-2a, II-2c, III-2a
- F-Switch: I-3, I-3a, II-3, II-3a, III-3, III-3a
- Load Chop: I-4
- Mapping OTF: II-1

For each of these groups, we introduced a corresponding group in the PipelineConfig.xml file in the `herschel.hifi.pipeline.generic`-package.

5.1.5. Some Details on Spectrum Data

Spectral Segment Data:

The spectral segment data is referred to as the part of the table datasets that contains on a per channel basis

- flux/intensity information,
- frequency/wave information (or suitable other equivalent unit),
- flags indicating possible problems with specific data points
- weights that give an indication on the statistical significance of specific data points.

We assume that, before running the generic pipeline, the instrument-specific pipelines (WBS/HRS) have successfully been processed. This means that the frequency calibration has been carried through (so that the columns containing the frequencies are filled accordingly) and some flags may have already been set (see, for instance, the WBS pipeline). The weights will be introduced as part of the generic pipeline (see the module DoChannelWeights below).

For all the tasks processed in the generic pipeline, we need to describe the rules adopted when processing and combining these data elements.

In particular, we need to specify both,

- the rules for how flags and weights are considered when processing the frequency and intensity information (e.g. use flags for filtering the data to be processed by restricting to unflagged channels), and
- how flags and weights are propagated, i.e. how flags and weights are defined for the output spectra (e.g. flags may be proagated by referring to a bitwise OR or AND arithmetic).

Known (or proposed but not yet implemented) flag values are summarized in the table below.

Value	Meaning
0 = 00000	Good
1 = 00001	Bad pixel
2 = 00010	Saturated

4 = 00100	Not observed
8 = 01000	Not calibrated
16 = 10000	Stitched-out (overlap)
32 = 100000	Glitched
64 = 1000000	Dark Pixel
128 = 10000000	Spur Candidate

Remark: Flags are considered in the data processing of the generic pipeline - they are just propagated. Weights are considered in the MkOffSmooth-task and the DoAverage-task.

Housekeeping Data important for the Generic Pipeline:

The following housekeeping data are important when processing the generic pipeline:

- LO-frequency ("LoFrequency"): Generally, for grouping comparable spectra or to check and identify phases in the FSwitch observations.
- Chopper position ("Chopper"): To check and identify phases.
- Buffer ("buffer"): Alternative to check and identify phases.
- Observation time ("obs time"): For interpolating possibly drifting intensity scales (hot/cold measurements) or "background" obtained by blank sky measurements.
- Hot/Cold load temperature("hot_cold"): Used in the intensity calibration (determination of the bandpass and the system temperature).

5.1.6. Initialization of Chopper Positions

The chopper voltages that can be found in the datasets (column "Chopper") included in the `HifiTimelineProduct` can be translated to chopper positions. This translation is specific for the different detector bands. The information on what chopper voltage ranges translate should be mapped to what position has been added to the calibration tree. In a preliminary step of the pipeline processing (level 1 and level 2) these chopper positions should be initialized by loading the corresponding tables from the calibration tree. This can be achieved with

```
from herschel.hifi.pipeline.generic.utils import ChopperPosition
ChopperPosition.initialize(calnode_generic, startDate)
```

where `calnode_generic` can be retrieved from the observation context by

```
calnode_generic= obs.calibration.getCalNode("downlink").getCalNode("generic")
```

and the `startDate` is the start date of the observation under consideration:

```
startDate= htp.meta["startDate"].value
```

Note that the start date is needed here since these chopper positions tables may be change over the life time of the instrument.

In case this step is omitted defaults are loaded. These can be inspected by looking at the table

```
defaults = herschel.hifi.pipeline.generic.cal.ChopperPositionsTable()
```

5.1.7. Pipeline Modules

The processing in the generic pipeline is independent of what backend has been used for the measurements (Wbs-H/V or Hrs-H/V). However, the generic pipeline processing depends on the particular AOT / observing mode that has been selected to do the measurement.

Some minor dependencies result due to small differences in the data produced during the instrument pipelines. As an example, we mention the "integration time" which is a double quantity for WBS and a series of double values for HRS. For HRS, different integration times are reported for the different segments while in the case of WBS, the same integration time results for all the sub-bands. The integration may be used when computing the weights quantities (see DoChannelWeights below).

For all the processing steps described below we assume that the WBS-/HRS-specific pipelines have **successfully** been processed. In particular, this means that the frequency columns are filled and the needed housekeeping data is available.

5.2. Level 1 Pipeline

5.2.1. CheckDataStructure

- **Purpose:**

  This module carries through a number of checks on the input product of type `HifiTimeline-Product` whether some important pre-conditions on the its structure and data items included are met.

  ```
  htp=checkDataStructure(htp=htp)
  ```

- **Description:**

  - *Assumptions:*

    - In the datasets of type "science" (the metadata element "sds_type"), the columns `bbtype` and `bbnumber` are available.

    - For adding the `isLine`-column to the summary table the bbtype is inspected. Here, only the bbtypes listed in the observing modes table (see above) give a sound distinction between ON (`isLine=True`) and OFF (`isLine=False`).

  - *Mathematics:*None

  - *Algorithms:*None

    - Uniform bbtype: If rows with different `bbid`'s are found within a dataset, these datasets are split up in several datasets. The `bbid` is computed from the `bbtype` and the `bbnumber` (see the corresponding columns) using the formula

      ```
      bbid = bbtype * 65536 + bbnumber
      ```

      For the split, the temporal order of the scans (observation time) is preserved so that only subsequent rows with identical `bbid` are included in the same dataset.

      Therefore, it ensures that each dataset corresponds to a particular phase of the observation (ON or OFF position, fixed LO settings). Note that in later versions of the software (at least 1.0 and after) this split is already accounted for when the timeline product is generated.

    - The "instrument" metadata element is also set at the dataset level to "HIFI". It ensures that "HIFI"-specific behavior is loaded when applying spectrum tools to these datasets.

    - Summary table: Adjust the summary table in case some datasets have been split since non-unique `bbid`'s have been found. The `isLine`-column entry is obtained by passing the bbtype to the isLine-method of the `CalBbid`-class in the package `herschel.hifi.pipeline.product` (which refers to the `BbidTable`-class. If not yet available, `LoFrequency` is added to the summary table.

- The frequency model is set to `null`.
- The "Band" metadata element that typically is found at the dataset level is also set at the level of the timeline product.
- For OBS-versions previous to 6.3 and FastDBS modes the chopper positions found for buffer A and buffer B are swapped.

- *Calibration Inputs:* None

- **Result:**

  If the the datasets need to be split a new htp is created - otherwise, the same htp (but with suitably adjusted summary table) is returned.

  - `HifiTimelineProduct` with "science" datasets that have a uniform `bbid`. The extended summary table reflects the change:

  Figure 5.1. Summary Table before Applying the Module

  Figure 5.2. Summary Table after Applying the Module

  - *Metadata* element "instrument" set to "HIFI".
  - *Columns:* Unchanged - datasets are split along the 'vertical' direction (along the observation time axis) so that the resulting datasets contain exactly the same columns.
  - *Errors and Warnings:*
    - Warning if no "obsMode" metadata element is found in the `htp`. Furthermore, a warning is produced if the observing mode is not listed in the xml-configuration file or cannot be identified by matching a suitable pattern. A suitable quality flag is generated (`isObservingModeFlagSet`).
    - In case an exception occurs the stack trace is written to the log - but the exception is caught so that the attempt can be made to continue the pipeline processing.

- **Remarks:**
  - Further data consistency checks are analyzed in the modules 'CheckPhases' and CheckFreqGrid (see below).

- **Questions, Issues:**
  - Should an additional check be included for whether HRS / WBS pipelines have been processed.
  - The splitting of the datasets would be obsolete when solving the Hifi-spr-1625.

5.2.2. CheckFreqGrid

- **Purpose:**

  This module builds groups of datasets - each being characterized with a fixed LO tuning. For each group, information is gathered on potential drifts in the frequency scale. This information is summarized in a suitable 'calibration' product of type `FreqRanges`. Finally, it adds the information on the LO frequency and the LO throw (for FSwitch observations) to the meta data of the datasets included in the timeline product.

```
freqRanges = checkFreqGrid(htp=htp, params=config)
```

- **Description:**
  - *Assumptions:*
    - For the analysis of the frequency drift it is sufficient to consider the "comb"-datasets. Since HRS data do not have comb data the frequency drift is trivial for HRS data.
    - All spectra that belong to the same group (i.e. have comparable LO-frequencies) are sub-sequent in time.
    - Within the group of datasets with comparable LO-frequencies it is assumed that all the spectra have the same number of bands and the same number of channels per band. The "comb" datasets, not necessarily need to have the same number of channels as the "hc" and the "science" datasets.
    - For FSwitch observations, two LO frequencies are found in the data. We consider the possibility that for comb data, only one of the two frequencies may be available.
    - All the spectra included in one dataset belong to the same group.
    - LoFrequency is given in GHz, the frequency / wave scale is given at the IF scale - i.e. in MHz.
  - *Mathematics:*
    - Datasets are defined to belong to the same LO-frequency group if the LO-frequencies are up to within a given tolerance the same. This tolerance is specified system-wide and set to 1 MHz. For each group, a unique integer id is assigned. This group id is stored in the metadata of the dataset (meta data element "frequencyGroup").
    - From a calibration point of view each frequency group should needs information for the intensity calibration (hot/cold load data) and for the off subtraction (if applicable). However, not all the observing modes collect these calibration inputs for all the LO tunings. In these cases, OFF- and hot/cold-data is shared across different LO groups. For the LO tuning groups that do not contain OFF or hot/cold-data a 'central' frequency group is associated that contains the needed information. This central LO group is selected by as the one closest in time.

      In addition to the `frequencyGroup` meta data element two additional fields are added here:
      - `frequencyGroup_OFF`: The associated central frequency for the off calibration.
      - `frequencyGroup_HC`: The associated central frequency group for the intensity calibration.
    - The quantity to measure drift between sub-sequent comb-measurements is defined through the formula

      ```
                          drift = max_subbands(abs(avg(w_t+1 -- w_t))) -/
      (obstime_t+1 -- obstime_t)
      ```

      where the frequency vector `w` is given in SI units and the `obstime` in seconds. Once this drift quantity exceeds a given tolerance, a warning is produced. This tolerance can be specified with the parameter `tolerance`. By default, this tolerance is set to 5 Hz / sec.
    - `FreqRanges`-product (see `herschel.hifi.pipeline.generic`): For each LO-frequency group, a dataset is included in the product. These datasets can be accessed through the key given by the group id. The datasets contain

    - the observation time of the comb spectra
    - the per channel frequencies of the comb measurements (per sub-band)
    - the drift measure for subsequent comb measurements (hence zero for the first row!).
    - the keys of the datasets in the original timeline product that belong to the given group (as `Int1d` metadata with key datasets").
    - minimum and maximum observation time of the frequency range group (metadata keys "startObsTime" and "endObsTime").
  - The `FreqRanges`-product contains utility methods that provide information on the drift between two observation times:
    - `getDistance(Long obsTime1, Long obsTime2)`: Defined as the integral of the (piece-wise constant) drifts between the two observation times. Provides the result in units of Hz.
    - `isComparable(Long obsTime1, Long obsTime2)`: Returns true if the two observation times fall into the same LO frequency group. Note that the we generally assume that datasets belongig to the same frequency group are subsequent in time.
    - `isComparable(int datasetIndex1, int datasetIndex2)`: Returns true if the datasets associated with the indices belong to the same LO frequency group.

    Note that the first method can be used to test whether frequency resampling needs to be considered by comparing the resulting distance with a given tolerance.
- *Algorithms:*
  - See above.
- *Calibration Inputs:*
  - Tolerance: Use the keyword `tolerance` to set a specific value (in SI units ie Hz/sec). In case drift values exceeding this tolerance level are observed a warning is created.

- **Result:**
  - Metadata:
    - In the datasets of the original htp, a meta data item `frequencyGroup` is set with the id (a String representing an integer) of the frequency range group the dataset belongs to. Only dataset types "science", "hc", "comb" are considered here.

      Similarly, if applicable, the meta data items `frequencyGroup_OFF` and `frequencyGroup_HC` are set.
    - The `loFrequency` is added to the meta data of the datasets. For F-switch observations, it is the LO frequency of the 'source', i.e. by default the LO frequency the sequence of scans within a dataset starts with. In addition, the `loThrow` is specified as the difference between the two LO frequencies found in the data (`loThrow = loSource-loRef`).
  - Flags: A quality flag is raised here in case the drift observed in the successive comb measurements exceed the tolerance (`isFrequencyDriftFlagSet`). Furthermore, a quality flag is raised if the frequency grouping fails (`isFrequencyChecksFlagSet`).
  - Calibration Outputs: The `FreqRanges` as described above.

Figure 5.3. A dataset of the `FreqRanges`-product corresponding to the single group.

- Errors and Warnings:
  - A warning is created if the calculated frequency drift measure exceeds the drift `tolerance` either specified by the user or the 5 Hz/sec taken as default.

- **Remarks:**
  - Since there are no comb spectra for HRS we will have zero drift here. The tables included per each group in the `FreqRanges` product is empty.
  - Note that the `FreqRanges` is intimately associated with the underlying timeline product. Therefore, the user should make sure not to use the methods `getDistance(...)`and `isComparable(...)` for different htp's.
  - Caution is needed when using the "datasets" metadata in the `FreqRanges`-product: These dataset id's are not updated when the structure of the timeline product is changed such as when datasets are removed or added. It makes reference to the timeline product obtained after applaying the CheckDataStructure-task.
- **Questions / Issues:**
  - Should all the data in the spectral segments be used when computing the drift measures? Should effects at the edges be masked out?

5.2.3. CheckPhases

- **Purpose:**

  Checks the patterns the chopper position, the buffer, the LO frequency and, for (Fast)DBS modes, the pointing follow within the hot-cold and the science datasets are followed.

```
phases = checkPhases(htp=htp)
```

- **Description:**
  - Assumptions:
    - The necessary housekeeping data is included in the datasets (chopper, buffer, LO frequency). The pointing checks are carried through once pointing information is available.
    - The evaluation of the patterns is done on a per dataset basis. We can expect meaningful results only if the datasets have been split up in accordance with the `bbid` (see `CheckDatastructure`-module).
    - For creating the log statements, the observing mode must be known (see table with the observing modes above).
  - Mathematics:

    The task is processed in two steps. First, the datasets of type "science" and "hc" are analyzed for whether the sequence of chopper, buffer and LO frequency values follow the patterns ABBA, ABAB, CONST. This step does not require any observing mode information. In a second step, the findings are compared with what would be expected for a given known observing mode and suitable log messages are generated.

- The result of the first step is a product of type `CalPhases` which contains an extended summary table for the underlying timeline product. As in the htp-summary table, the information for each dataset is filled in a separate line. For each of the three quantities we have three columns with

  - the pattern ("ABBA", "ABAB", "CONST", "NONE") - variations are "AB" if only two lines are available so that there is no distinction between ABBA and ABAB;

  - the values (as a comma-separated list);

  - the initial value.

  For the buffer and the LO frequency, the values are tested for obeying the patterns within suitable tolerances. For the LO frequency this is set to 1MHz, for the chopper position the ranges are used as specified in the ChopperPosition table (see below). For the pointing information (in the FasDBS-modes), we allow for a tolerance of 2.5 arc secs for RA and DEC.

Figure 5.4. The summary table included in the `CalPhases`-product.

You can inspect the current (band-specific) values used by the system by calling e.g.

```
             from herschel.hifi.pipeline.generic.utils import
ChopperPosition
             band = 1 # the number of the detector band
             print ChopperPosition.COLD.getLowerBound(band)
             print ChopperPosition.COLD.getPosition(band)
             print ChopperPosition.COLD.getUpperBound(band)
```

and similarly for `HOT`, `LEFT`, `CENTER` and `RIGHT`.

- Checking for consistency with the given observing mode - or, strictly speaking, the observing mode groups. The actual values are listed in the table but not tested for consistency.

  - Position Switch and MappingOTF: Constant patterns for Chopper, buffer and LO frequency in the "science" datasets, ABBA pattern for "hc" datasets in the Chopper and the buffer.

  - DBS: ABBA patterns for Chopper and buffer in both, "science" and "hc" datasets. CONST for LO frequency.

  - Fast DBS: ABAB patterns for Chopper and buffer in "science" datasets, ABBA for Chopper and buffer in "hc" datasets. CONST for LO frequency.

  - FSwitch: ABBA patterns for LO frequency and buffer while CONST for Chopper in "science" datasets. ABBA patterns for Chopper and buffer while NONE for LO frequency in "hc" datasets.

  - LoadChop: ABBA patterns for Chopper and buffer while CONST for LO frequency in "science" and "hc" datasets.

- Algorithms: see above.

- Calibration input: None.

- **Result:**

  - The original product is not changed.

- Flags: Are created in case inconsistencies between the expected and the observed patterns for chopper, buffer and LO frequency are found and in case the number of distinct values .
  - Chopper: `isChopperPatternFlagSet` if an inconsistent chopper pattern is found and `isChopperValuesFlagSet` if the number of different chopper values (per dataset) is not as expected.
  - Buffer: `isBufferPatternFlagSet` if an inconsistent buffer pattern is found and `isBUfferValuesFlagSet` if the number of different buffer values (per dataset) is not as expected.
  - LO frequency: `isLofPatternFlagSet` if an inconsistent pattern in the LO frequencies are found and `isLofValuesFlagSet` if the number of different LO frequencies (per dataset) is not as expected.

  Furthermore, a flag is raised in case the `ChcekPhases`-task could not be finished (`isPhaseChecksFlagSet`).
- Calibration Outputs: The `CalPhases`-product as described above.
- Errors and Warnings:
  - Log warnings in case patterns or the number of distinct values are found which are not 'as expected' (see flags above).

- **Remarks, Issues:**
  - ...

5.2.4. MkFluxHotCold

- **Purpose:**

  Hot/Cold load measurements are used to obtain both, the receiver (or system) temperature and the bandpass. These quantities are computed for each "hc" dataset in the original timeline product and included in a calibration product (of type `CalFluxHotCold`). Bandpass is used for the intensity calibration and the system temperature, possibly, for the determination of the channel-dependent weights to be included in the spectra (see the module `DoChannelWeights`).

```
calHC = mkFluxHotCold(htp=htp, params=config, coupling=coupCoeff)
```

  where the `CoupCoeff` are the coupling coefficients which are typically retrieved from the calibration tree.

  In the pipeline, checking for differences in the mixer currents exceeding a given tolerance is done - if it is exceeded a row flag is set. The tolerances are specified by

```
calHC = mkFluxHotCold(htp=htp, params=config, coupling=coupCoeff,
validatorTolerance=0.02)
```

  or

```
calHC = mkFluxHotCold(htp=htp, params=config, coupling=coupCoeff,
validatorTolerance=calTable)
```

  where the calTable is of type `GenericPipelineCalProduct` that is typically retrieved from the calibration tree

```
calTable =
obs.calibration.getCalNode("downlink").getCalNode("generic").getProduct("mixerCurrentTolerances")
```

- **Description:**
  - Assumptions:
    - Sufficient hot/cold measurements are available in the observation. To be more specific, this means: For each LO setting, at least one hot/cold dataset (metadata element 'sds_type' set to 'hc') is available. Within each of those datasets, both, hot and cold measurements can be found. For FSwitch observations, each 'hc' dataset should contain hot and cold load data at each of the LO frequencies (separated by the LO throw).
    - The hot and cold scans within the 'hc' datasets can uniquely be identified by one of the following indicators: Chopper position, buffer value or assuming a pattern (ABBA or ABAB).
    - Frequency drift of the scans within a single 'hc' dataset is not considered. As a result, the hot and cold spectra found within a single 'hc' dataset can be combined (as described below) without frequency re-sampling.
  - Mathematics:
    - Consecutive *hot/cold datasets* that have consistent LO frequencies are (ie belong to the same frequency group, see the `CheckFreqGrid`) are *merged*.
    - From the hc dataset obtained in the previous step, the *hot and cold load scans are identified*. As mentioned above, three schemes are available to do this. Specify the string-parameter `indicator` to one of the following values:
      - 'Chopper': the hot/cold scans are identified from the chopper position - have a look at the table with the chopper position as described in section `CheckPhases`! Using the chopper value as the indicator is the default.
      - 'buffer': The buffer value = 1 is associated with hot, all the other with cold.
      - 'pattern': When choosing this indicator, by default, a 'ABBA' pattern is assumed - starting with hot. An alternative would be a pattern of the form 'ABAB'. This option can be selected by setting the parameter `isABBA = False`.

      as a result of this identification, a `Bool1d`-column is added to the dataset with name `isHot`.
    - Within each hot/cold dataset, *all the hot* (with the same LO frequency) and *all the cold* with the same LO frequency are *averaged*. To be specific, the following operations are carried through:
      - Operations on the spectral segment data:
        - `flux`: Simple average.
        - `weight`: Typically, not yet available at this stage.
        - `flag`: Flags are considered by ignoring the flagged values or propagated (Bitwise OR) in case no unflagged values exist (per channel)
        - `wave/frequency`: Simple average
      - Other columns:
        - `Chopper`: Average
        - `LoFrequency`: Average
        - `hot_cold` (temperature of hot /cold load: T_h,T_c): Average
        - `integration time` (WBS: scancount / HRS:): sum

- `scancount` (WBS): sum
- `packet time`: min
- `obsTime`: Average

- Once these averages are available, the receiver temperature and bandpass can be calculated using the formulas:
  - Receiver temperature:

```
                              [(eta_h+Y*eta_c-Y)*j_h -- (eta_h
+Y*eta_c-1)*j_c] -/ [Y-1]
```

    where `Y` denotes the ratio of the average hot and the average cold flux data (`Y=avg_h/avg_c`) and `j_h` and `j_c` denote the the thermal radiation fields temperatures at temperature `T_h` and `T_c`. These temperatures are defined as functions of the LO frequency.

  - Bandpass:

```
                              (avg_h-avg_c)/(eta_h + eta_c -- 1)*(j_h -- j_c)
```

    In the formulas above, the average hot and cold load flux (`avg_h` and `avg_c`) are densities (flux per wave scale unit).

  - In general, the coupling coefficients `eta_h` and `eta_c` are treated as quantities dependent on LO frequency and band. These coefficients are looked up from suitable tables passed by a calibration product to the task (task parameter `coupling`). In case no band information is found in the data band "1a" is assumed.

- Algorithms: see above
- Calibration inputs:
  - Coupling coefficients `eta_h` and `eta_c` provided in form of the product `GenericPipelineCalProduct`; use the parameter name `coupling`.

    Running the task from within the pipeline, the coefficients are obtained from the calibration tree. Within the observation context (`obs`) you can find these coefficients for H-polarization by

```
coupCoeff=obs.calibration.getCalNode("downlink").getCalNode("generic").getProduct("couplingEf
H")
```

    and similarly for V-polarization.

- **Result:**
  - Input timeline product: only the 'hc' datasets are processed by this module.
    - Meta data: -
    - Columns: `isHot` added
    - Flags: Quality flag if not sufficient hot and cold data are found in the hot/cold datasets (`isHotColdDataFlagSet`). The flag `isTSysFlagSet` is raised if the task could not be finished (eg because the hot and cold radiation field temperatures are nearly identical).

- Calibration output: Calibration product of type `CalFluxHotCold` (subclass of a `HifiProduct`). It is returned by the module or can be accessed by the key `cal`. This product contains bandpass and receiver temperature possibly at different observation times and LO frequencies. Bandpass and receiver temperature are stored in different datasets and For different LO frequencies different datasets are used. Bandpass datasets have even integer keys, receiver temperatures odd integer keys. Hence, the system temperature for the LO frequency found in the data is obtained e.g. by

```
tsys1 = calHC[1]
```

  - Meta data: Most of the metadata of the original timeline product are copied.
  - Summary table: Similar to the summary table of the timeline product
    - `dataset`: The integer key used within the product.
    - `type`: 'rec temp' or 'band pass'
    - `LoFrequency`: the LO frequency associated with the given rec temperature and bandpass.

Figure 5.5. Summary Table after Applying the Module

  - Datasets (of type `HifiSpectrumDataset`):
    - The receiver temperatures and the bandpass are in different datasets and the results associated with different LO settings are also included in different datasets. Per 'hc' dataset of the original timeline product one bandpass and one receiver temperature spectrum is created as one row in a bandpass and receiver temperature dataset are created (in case of FSwitch observing modes such spectra are crated in two different bandpass and receiver temperature datasets). Each row in given dataset is associated with a specific observation time. The columns in the original 'hc' datasets are processed following the rules specified for the average above - except for the flux which is calculated according to the formulas given above.
    - The product provides utility methods to access the bandpass and system temperatures that correspond to given science datasets:
      - `getRecTempCalibration(loGroup, loFreq)`
      - `getBandpassCalibration(loGroup, loFreq)`
        In case the `loGroup` (the LO frequency group assigned in the `CheckFreqGrid`-task) is missing, the receiver temperature or bandpass calibration with the closest LO frequency is returned.
  - Errors and Warnings:
    - Warning: *Available hot/cold data is not sufficient to build a calibration.* if not both, hot and cold measurements are available.
    - Warning: If the number of LO frequencies in a given 'hc' dataset is not consistent with the observing mode.
    - Caught exception: In-distinguishable radiation field temperatures (j_h-j_c < 10^-8K)

- **Remarks:**

- Note that for WBS data the system temperature (and the bandpass) is computed only for the restricted subband ranges with the subband edges cut during the WBS pipeline processing.

- **Questions, Issues:**

  - Add documentation on the validation of the mixer current differences, how the tolerance level can be specified by either passing a double number or a suitable calibration product of type `GenericPipelineCalProduct`

5.2.5. DoChannelWeights

- **Purpose:**

  Fill values into the weights column (spectral segment data) by making use of the receiver temperature (from CalFluxHotCold calibration product obtained after processing MkFluxHotCold) and/or the integration time or using a statistical estimator for the sample variance of the spectra within a given (moving window). In addition, a smoothing scheme can be selected to smooth the weights obtained in the first step. Note that these weights are also a measure for the resolution (resolution is inversely proportional to the square root of the weights). The idea behind the weights is that they can be referred to when combining different scans (e.g. calculating an average).

```
doChannelWeights(htp=htp, cal=hc, params=params)
```

- **Description:**

  - Assumptions:

    - Different formulas are implemented that are used for the calculation of the weights. It is assumed that depending on the formula selected the required input data is available / provided as input to the module. As an example, we mention the hot/cold calibration data of type `CalFluxHotCold` which is computed in the `MkFluxHotCold` module. It is used when the radiometric formula is used for the definition of the weights.

  - Mathematics:

    Different formulas are available that can be selected by setting the parameter with key `definition` to

    - `definition='integrTime'`: Define the weights equal to the integration time (independent of the channel).

      ```
      w = t_integr.
      ```

    - `definition='variance'`: Define the weights equal to the inverse of the sample variance for a moving window within the scan. Here, an additional parameter, the window width, is needed (`width`).

      ```
      w_k = 1 -/ v_k, v_k = sum_j=k-r,...,k+r (x_j -- m_k)^2 -/ (2r+1).
      ```

    - `definition='radiometric'`: Define the weights equal to the inverse of the of the resolution given by the radiometric formula, i.e.

      ```
      w = t_integr -/ t_sys^2.
      ```

      In addition, the result can be smoothed by a Gaussian or a box car filter over a width specified by the `width`-parameter. Use the keyword `smoothing` to set a filter (`smoothing='Gaussian'`, `smoothing='box'`). Finally, to obtain the system temper-

ature for a given observation time, an interpolation scheme needs to be adopted (set the keyword `interpolator` to `'PREVIOUS'`, `'LINEAR'`, `'NEAREST'`, `'SPLINE'`).

- Algorithms: see above.
- Calibration input: Calibration product of type `CalFluxHotCold` resulting from `MkFluxHotCold` (use key `cal`).

- **Result:**
  - Datasets: Weights columns are added to the datasets of type 'science' included in the htp.
  - Flags: Quality flag is raised if the task could not successfully be completed (`isChannelWeightsFlagSet`).
  - Errors, Warnings
    - Caught exception in case a required input is not provided as input to the module.
- **Remarks:**
  - In operations where the channel width is changed (such as when re-sampling the flux to a different frequency scale) the weights need to be adjusted since the resolution is changed.
  - Setting the keyword `ignore` to `True`, the calculations configured for this module are skipped.
- **Questions, Issues:**
  - Radiometric formula: Incoporate suitable coefficients to give the weights a physical meaning. Include resolution (~channel width).
  - Sample variance in presence of lines, normalization of the weights...
  - Add checks for the consistent frequency ranges?

5.2.6. DoRefSubtract

- **Purpose:**

  Reference measurements taken from blank sky (in DBS modes), from an internal load (in LoadChop modes) or taken at a different LO frequency (in FSwitch modes) are subtracted from the source measurements in order to eliminate instrumental drifts from the source measurements (drift of the overall system response). It constitutes one part of the double subtraction scheme typical for HIFI. With the `ignore` paraneter set to `True` the execution of the task can be skipped.

```
doRefSubtract(htp=htp, params=params)


doRefSubtract(htp=htp, indicator='pattern', isABBA=True, startsWithRef=True,
offStartsWithOpposite=True)
```

- **Description:**
  - Assumptions:
    - It is assumed that the frequency scales of the pairs of spectra that are subtracted from each other are close so that no frequency re-sampling of the reference spectra is necessary.

- For position switch reference observing modes, the subtraction of the reference position is treated as an OFF subtraction.

- Mathematics:

  - Take the difference between subsequent source ('source') and reference measurements ('ref') included in datasets of type 'science'.

  - Datasets of a type different than 'science' are not processed.

  - Within the science datasets, the source and reference measurements are identified either by assuming a specific pattern (see `CheckPhases` above) - or by looking up specific chopper, buffer or LO frequency values. The method adopted to identify 'source' and 'ref' is specified by the parameter `indicator`:

    - `indicator='pattern'`: Assume a pattern (specific to the observing mode). The pattern is selected by specifying the parameter `isABBA` to `True` or `False`. Using the parameter `startWithRef` you can specify whether the sequence starts with a 'ref'-(`startsWithRef=True`) or with a 'source'-measurement (`startsWithRef=False`). For (Fast-)DBS modes, the role of 'source' and 'ref' change when going from ON to OFF measurements. This can be expressed by the parameter `offStartsWithOpposite`.

      Typically, the following settings hold true for the different observing modes:

      - Fast Chop DBS: ABAB pattern starting with A=ref in the ON datasets (isLine=true) and with A=ref in the OFF datasets - hence `isABBA=False, startsWithRef=True, offStartsWithOpposite=True`.

      - Slow Chop DBS: ABBA pattern starting with A=ref in the ON datasets (isLine=true) and with A=source in the OFF datasets (isLine=false) - hence `isABBA=True, startsWithRef=True, offStartsWithOpposite=True`.

      - Position Switch: Reference subtraction ignored here.

      - Frequency Switch: ABBA pattern starting with A=source - hence `isABBA=True, startsWithRef=False, offStartsWithOpposite=False`.

      - Load Chop: ABBA pattern starting with A=ref - hence `isABBA=True, startsWithRef=True, offStartsWithOpposite=False`.

    - `indicator='buffer'`: When using 'buffer' as an indicator the values to be associated with 'source' and 'ref' should be specified. Here, the parameter `phaseValues` should be set.

      ```
      phaseValues = {'source':2, -'ref':1}
      ```

    - `indicator='Chopper'`: When using 'Chopper' as an indicator the values to be associated with 'source' and 'ref' should be specified. Here, the parameter `phaseValues` should be set.

      ```
      phaseValues = {'source':-4.1, -'ref':-7.0}
      ```

      Alternatively, you can specify the chopper positions as a string with capital letters such as

      ```
      phaseValues =
{'source':'CENTER', -'ref':'LEFT'}
      ```

- `indicator='LoFrequency'`: When using 'LoFrequency' as an indicator the values to be associated with 'source' and 'ref' should be specified. Here, the parameter `phaseValues` should be set.

```
phaseValues = {'source':600.00, -'ref':600.24}
```

(in GHz)

- Operations on spectral segment data:
  - Flux: Simple Difference
  - Flag: Bitwise OR
  - Weight: Weight of the 'source'.
  - Wave: Taken as the wave-data of 'source'.
- Operations on other columns: For all the other columns, the values of the 'source' phase are copied. An exception will be the `rowflag`
  .

- Algorithms:
  - see Mathematics above.
  - Calibration input: None

- **Result:**

  The science datasets are replaced by new datasets that contain the difference spectra. As a result, the number of scans in the science datasets is reduced by a factor of two. All the other datasets remain unaffected.

  - Meta data: `isRefSubtractFlagSet` added to the meta data.
  - Columns: For all columns not included in the 'spectral segment' part of the datasets, the data is simply copied from the 'source' positions of the original datasets.
  - Flags: `isRefSubtractFlagSet` in case something goes wrong with the identification of the phases.
  - Calibration Outputs: None
  - Errors and Warnings:
    - Warning in case datasets are found with number of rows smaller than 2.

- **Remarks:**
  - For the Frequency Switch modes, the shifted and the un-shifted spectra are overlayed after the DoRefSubtract (with opposite signs). These spectra need to be 'folded'. See the module DoFold in the same package.

- **Questions / Issues:**
  - Propagation of the flags not yet implemented? Currently, only the flag of the source is copied to the result.

- Similarly, the weights are defined as the weights of the source. How should the weights be defined?
- Add documentation on the validation of the mixer current differences, how the tolerance level can be specified by either passing a double number or a suitable calibration product of type `GenericPipelineCalProduct`. Approach is identical to MkFluxHotCold - see there for further details.

5.2.7. MkOffSmooth

- **Purpose:**

  Average and smooth (on the frequency scale) the flux data from the OFF measurements. The module is processed on a per dataset basis. This means that for each OFF dataset (of type 'science') a baseline is constructed and included in a product of type `CalOffBaseline`. These baselines will be subtracted in the `DoOffSubtract` module. Note that the construction of this OFF baseline is applicable only for frequency switch, load chop and mapping OTF observing modes. In case no OFF measurements are found in the data no output is produced. With the `ignore` parameter set to `True` the execution of the task can be skipped.

```
baseline = mkOffSmooth(htp = htp, params = params)
```

```
baseline = mkOffSmooth(htp = htp, filter='Gaussian', width=20)
```

- **Description:**
  - Assumptions:
    - The module `DoRefSubtract` should have been processed successfully - otherwise, 'source' and 'ref' scans would need to be distinguished and treated separately.
    - It is assumed that potential drifts on the frequency scale can be neglected when doing the average. I.e. we assume that frequency re-sampling is not necessary.
  - Mathematics:
    - Identify the OFF datasets (with type 'science') and process them on a per dataset basis. The filtering or fitting is done on a individual per segment basis.
    - Modes: Different modes are available for configuring the computations. Use the parameter `mode`:
      - `mode='filter'` (default): First take the average over all the spectra included in the OFF dataset. Second apply a suitable convolution kernel (filter) to do the smoothing. You can configure the type of smoothing using the parameter 'filter'. For the time being, a Gaussian and Boxcar filter are possible choices - hence, `filter='Gaussian'` (default) or `filter='box'`. The (integer) width of the smoothing kernel (the number of channels) can be set via the parameter `width`. In case no width is specified, the width is calculated from

        ```
        width = Number of ON Scans -/ Number of OFF Scans
        ```

        For the box car filter, it corresponds to the number of channels to be included in the smoothing; for the Gaussian filter, it corresponds to the standard deviation of the Gaussian.

- `mode='fitter'`: First take the average over all the spectra included in the OFF dataset. Second apply a suitable fitter function to the averaged spectrum - at the moment, only polynomial fitting can be configured by specifying the degree of the polynomial to be applied using the parameter `degree_polyomial`.
- `mode='avg'`: Just take the average over all the spectra included in the OFF dataset.

- Operations when taking the average:
  - Flux: Weighted average. Use weights as included in the spectral segment data. If no weights are available, do a simple arithmetic average. Only consider unflagged channels - in case only flagged channels are available calculate the average for (all) the flagged channels available and propagate the flag values.
  - Flag: Bitwise AND if unflagged channels are available - bitwise OR otherwise.
  - Weight: The sum of the weights of the individual scans. Note that this is consistent with the sum of the variances once the weights are inversely proportional to the variances in accordance with minimum variance optimization.
  - Wave: Take the wave of the first scan found in the dataset.
  - Operations on other columns:
    - Chopper: (Arithmetic) Average
    - LoFrequency: (Arithmetic) Average
    - integration time: Sum
    - obs time: (simple) Average
    - packet time: Minimum
- Operations when smoothing:
  - Flux: Apply the filter / fitter within a moving window of the original flux values. Select channels that are non-flagged. In case the selection for a given result channel is empty, compute a result including the flagged channels while propagating the flag values.

    To summarize, the calculation of the flux is of the form

    ```
                              flux_smoothed(n) = [sum_k weight(k)flux(k)
    Phi(n-k)] -/ [sum_k weight(k) Phi(n-k)]
    ```

    where flux and weights are restricted to the unflagged channels and Phi is the kernel of the filter.
  - Weight: Some resolution is lost, therefore, the weight should be decreased. This, however, depends on the particular smoothing / fitting model.

    ```
                              weight_smoothed(n) = [sum_k weight(k) Phi(n-
    k)] -/ width.
    ```
  - Flag: Propagate the flags in accordance with a bitwise AND logic. In case no unflagged channels are available, propagate according to a bitwise OR logic.

    - Wave: Copy the original values.
  - Algorithms:
    - See description above.
  - Calibration Input:
    - None
- **Result:**

  The timeline product is not transformed by this module. Rather, a calibration product of type `CalOffBaseline`is created that contains possibly several baselines - each covering a certain period of (observation) time defined by the original OFF dataset. Use the parameter `cal` to retrieve this product from the task. The baselines with common LO frequency are packed into the same dataset (of the same type as the original OFF dataset, i.e. `Wbs/HrsSpectrumDataset`).

  - Meta data: A copy of the meta data of the original timeline product.
  - Datasets: The datasets contain the same columns as the original OFF dataset with the flux/weight/flag columns processed as described above. The meta data is just copied from the first OFF dataset (in the timeline).
  - Flags: Quality flag raised if no off baseline could be calculated (`isNoOffBaseline-FlagSet`).
  - Errors and Warnings:
    - Warning if no OFF datasets are found in the data (e.g. for all the observing modes ending with 'NoRef') --> **TODO**.
    - Warning if no baseline can be provided by the module.
    - Warning if no valid width has been specified or computed as default (see the algorithm above) and the default 1 is used instead.
- **Remarks:**
  - For position switch and DBS modes, no smoothing nor an averaging is carried through.
    - For position switch: The datasets with the OFF measurements contain the same number of scans as the datasets with the ON measurements. Associated scans of the ON and the OFF datasets are subtracted from each other on an individual per scan basis.
    - For DBS: The difference scans calculated in the DoRefSubtract and included in the OFF datasets must neither be smoothed nor averaged since these contain signal information.
- **Questions / Issues:**
  - Is the way the weights and flags are included in the processing of average and smoothing ok?
  - Add rms of the original scans and of the baselines to the baseline data.

5.2.8. DoOffSubtract

- **Purpose:**

  Subtract the calibrated baseline(s) from the ON measurements (as in the load chop or frequency switch modes) or subtract the OFF from the ON scans on a row-by-row basis for the position switch

or average the ON and the OFF scans on a scan-by-scan basis for (Fast) DBS modes. With the `ignore` paraneter set to `True` the execution of the task can be skipped.

```
doOffSubtract(htp=htp, cal=baseline, params=params)
```

- **Description:**
  - Assumptions:
    - DoRefSubtract has been processed.
    - For the frequency switch and the load chop modes, the baselines are obtained from a `CalOff-Baseline`-product passed to the module using the signature keyword `cal`. Typically, the data is obtained from the MkOffSmooth module. If no baseline is available, no baseline subtraction is processed (see e.g. the modes with a label ending with 'NoRef'). The baselines and the spectra from which the baseline should to be subtracted from need to match in shape and frequnecy scale. No consistency check is done for that.
    - For the position switch and the DBS modes, no baselines are fed into the component. The information on the OFF position is directly obtained from the htp to be processed. No smoothing / averaging is involved here.
  - Mathematics:
    - A 'mode' parameter is specified which indicates whether the task should
      - Combine calibrated baselines with ON spectra by interpolating and subtracting (FSwitch and Load Chop observing modes): `mode="interpol"`
      - Subtract OFF spectra from ON spectra on a scan-by scan basis (Position Switch Reference observing modes): `mode="row-wise-subtract"`
      - Average On and OFF spectra on a scan-by scan basis (DBS and Fast DBS observing modes): `mode="row-wise-avg"`
    - The `interpol` mode can be applied only if baseline information is provided to the module (parameter `cal`). For each scan found in a ON dataset, a specific baseline is constructed by interpolating the baselines found in the HifiBaselineProduct to the observation time of the ON scan. For the interpolation, several schemes are available that can be configured by the user (keyword `interpol`): 'LINEAR', 'NEAREST', 'PREVIOUS', 'NEXT', 'CUBIC SPLINE'.
      - Operations on spectral segment data:
        - Flux: The interpolated flux value of the baseline is subtracted from the flux of the ON scan.
        - Flag: The flags in the baselines are propagated using bitwise OR to the baseline to be subtracted. The actual subtraction propagates the flags using a bitwise OR.
        - Weight: The weight of the result is defined as the weight of the ON scan.
        - Wave: The wave is set as the wave of the ON scan.
      - Operations on other columns:
        - All the other columns are just copied from the ON datasets.
    - The 'row-wise-subtract' mode is applied in combination with the Position Switch Reference mode: Here, On and OFF datasets need to have exactly the same number of rows. This allows

to subtract OFF scan from associated ON scans (identified by the row index) on a scan-by-scan basis. Conceptually, this subtraction scheme resembles rather a reference subtraction than an OFF subtraction. For practical reasons, it is treated here as an OFF subtraction.

- Operations on spectral segment data:
  - Flux: Subtract
  - Flag: Bitwise OR
  - Weight: Weight of the ON scan.
  - Wave: Wave of the ON scan.
- Operations on the other columns:
  - All the other columns are just copied from the ON datasets.

- For DBS and Fast DBS, the ON and OFF datasets occur in pairs with exactly the same number of scans per dataset (`mode="row-wise-avg"`). Basically, the information included in the ON and OFF datasets is the same, except that the optical paths are reversed between ON and OFF. By taking the average of ON and OFF scans on a per scan basis, part of the standing wave contribution can be eliminated.
  - Operations on spectral segment data:
    - Flux: Weighted average. Use weights as included in the spectral segment data. If no weights are available, do a simple arithmetic average.
    - Flag: Bitwise OR
    - Weight: Weight of the ON scans.
    - Wave: wave of 'source'.
  - Operations on other columns:
    - Chopper: (Arithmetic) Average
    - LoFrequency: (Arithmetic) Average
    - Integration time: Sum
    - obsTime: Average
    - packetTime: Average

- Algorithm:
  - Configuration possibilities as described above.
- Calibration inputs: Product containing the baseline(s).

- **Result:**
  - ???
  - Meta data:
  - Datasets: The results are stored in ON datasets (isLine=True). Any datasets labeled as OFF (isLine=False) are removed from the timeline product in the end.

- Columns: see the rules described above.
- Flags:
  - If the datasets to be subtracted or averaged (FastDBS, DBS) do not have the same size: `isOnOffPairSizeFlagSet`
  - If different numbers of ON and OFF datasets are found (DBS; FastDBS, PositionSwitching): `isOnOffProcessingFlagSet`
  - I f not for all ON datasets a baseline to subtract is found (load chop, f-switch): `isSubtractOffBaselineFlagSet`
- Calibration Outputs: None
- Errors and Warnings:

- **Remarks:**
  - ...
- **Questions, Issues::**
  - Is the way weights are treated ok? --> TODO
  - Unclear how to deal with drifting frequency scales re-sampling or not? At the moment, drifts are not checked and no re-sampling is done.
  - In particular it is not possible to plug in baselines from other observations and do a suitable resampling.
  - Add documentation on the validation of the mixer current differences, how the tolerance level can be specified by either passing a double number or a suitable calibration product of type `GenericPipelineCalProduct`. Approach is identical to MkFluxHotCold - see there for further details.

5.2.9. DoFluxHotCold

- **Purpose:**

  The calibrated intensity scale obtained in the `MkFluxHotCold` task - the bandpass - is applied to the flux data. This transforms the intensity scale to Kelvin units. All science data (dataset with type 'science') found in the given product are adjusted in this way. The calibration data is passed in form of `CalFluxHotCold` product to the module using the signature keyword 'cal'.

```
doFluxHotCold(htp = htp, cal = hc, params = params)
```

- **Description:**
  - Assumptions:
    - A calibration product of type `CalFluxHotCold` is provided to the component which contains the bandpass information. For each LO frequency found in the science data, one or several bandpass calibrations (possibly at several observation times) should be available. Two LO tunings are considered consistent if the match within a tolerance of 1MHz. Furthermore, the sub-band structure and the corresponding (IF) frequency scales should be consistent and no frequency resampling is necessary.
    - For F-Switch modes, a bandpass is available at both LO frequencies (separated by the LO throw). This makes it possible to consider calibration schemes in which the division of the flux

by the bandpass is carried through for each LO frequency separately, i.e. before the `DoRefSubtract`. The current calibration scheme applies the `DoFluxHotCold` after the `DoRefSubtract` and the bandpass with the same LO frequency as the source phase is used.

- The wave scales of the science data and the wave scales of the interpolated bandpass are assumed to be close so that resampling the bandpass to the frequency of the science scans is not necessary.

- Mathematics: The module iterates through all the scans included in the science datasets and processes them on an individual basis.
  - Interpolation: For each (science) scan, a bandpass with consistent LO frequency is retrieved from `CalFluxHotCold`. A suitable interpolation scheme is used to shift to the observation time of the science scan. The user can configure the interpolation scheme: Keyword: 'interp' with values 'LINEAR', 'NEAREST', 'PREVIOUS', 'NEXT', 'CUBIC SPLINE'
  - Division by the bandpass:
    - Operations on spectral segment data:
      - Flux: Divide, i.e. compute the ratio R = S/B ('S': science, 'B': bandpass).
      - Flag: Bitwise OR
      - Weight: Weight of the science scan (nominator).
      - Wave: Wave of the science scan (nominator).
    - Operations on data in other columns:
      - All values are taken from the nominator (the science data)
- Algorithms: see above.
- Calibration inputs: The input should be provided in form of a `CalFluxHotCold` product typically is obtained by the module `MkFluxHotCold`.

- **Result:**
  - Meta data: Nothing added except for quality flags (see below).
  - Datasets: All science datasets found in the product are transformed. The `flux`-columns in the datasets now have Kelvin
  - Flags: For the scans for which the bandpass calibration fails (because e.g. no suitable bandpass has been found) a row flag is set (once a row flag column is present). The flag is set to '2'. In addition a quality flag is raised: `isIntensityCalibrationFlagSet` (which can also be seen in the meta data).
  - Errors and Warnings:
    - Severe warning if for some scans no suitable bandpass is available.
    - Warning if for some scans more than two different LO frequencies are found. This would indicate that e.g. the `CheckDataStructure` has not been processed.
- **Remarks:**
  - ...
- **Questions, Issues:**

- Is the way the weights for the resulting scans are computed ok?.
- Add documentation on the validation of the mixer current differences, how the tolerance level can be specified by either passing a double number or a suitable calibration product of type `GenericPipelineCalProduct`. Approach is identical to MkFluxHotCold - see there for further details.

5.2.10. DoVelocityCorrection

- **Purpose:**

  Corrects the frequency scale for the velocity of the spacecraft and possibly of the source. In contrast to `doRadialVelocity` it uses a relativistic approach when correcting for the motion of the spacecraft relative to SSB or LSR or, in case of SSO's, relative to the SSO. For non-SSO's, the motion of the source relative to the LSR or the SSB is treated classically.

  Possible target rest frames to transform to (see parameter targetFrame) are "HSO" (short for Herschel Space Observatory), "GEOCENTRIC", "SSB" or "BARYCENTRIC", "LSR" or "SOURCE". By default, the task transforms to the "LSR" frame for non SSO's and "SOURCE" for SSO's.

  The velocity of the spacecraft can be provided in different ways. The one used in the pipeline is by retrieving the velocity information from the auxiliary context from the observation context:

```
                doVelocityCorrection(htp=htp, aux=obs.aux)
```

  In contrast to `doRadialVelocity`, a 3-vector velocities are needed for the relativistic correction.

  The task also allows to transform to the rest frame of the source even for non-SSO's. Here, additional 'velocity' information is needed which is no longer a 3-vector though but rather just a scalar parameter:

```
                doVelocityCorrection(htp=htp, targetFrame="SOURCE",
aux=obs.aux, velocity_source=20.0)
```

- **Description:**
  - Assumptions:
    - Wave scale of the spectra must not be expressed as velocities.
    - Pointing information should be included in data in form of columns "longitude" and "latitude".
  - Mathematics:
    - For the relativistic correction for the motion of the spacecraft relative to the earth, the sun (ssb) or LSR the following factors are applied:
      - From spacecraft to earth: `f(v_earth, p)/f(v_hso, p)` where `f(v,p) = (1+v*p/c)/sqrt(1-v*v/c^2)`, `v_earth` and `v_hso` are the 3-vectors of the earth or the spacecraft, respectively, relative to SSB in the SSB frame of reference and p is the unit vector in direction of the pointing in the SSB frame;
      - Similarly, from spacecraft to SSB: `1/f(v_hso, p)`;
      - Finally, from spacecraft to LSR: `f(v_lsr/f(v_hso, p)` where `v_hso` is the 3-vector of the LSR relative to SSB in the SSB frame of reference.

- The 3d velocity of the spacecraft can be provided in different ways:
  - from an object of type herschel.share.fltdyn.ephem.Ephemerides passed as 'ephem' parameter to the task:

    ```
    doVelocityCorrection(htp=htp, ephem=ephemerides)
    ```

  - from an auxiliary context passed as 'aux' parameter to the task:

    ```
    doVelocityCorrection(htp=htp, aux=obs.aux)
    ```

  - from the datasets to be processed, in a column named 'velocity_hso_1', 'velocity_hso_2', 'velocity_hso_3': that have previously been entered into the data:

    ```
    doVelocityCorrection(htp=htp)
    ```

- There are different options to parametrize the transformation to the rest frame of the source (which only apply if `targetFrame="SOURCE"`). Common to all these options is a radial redshift factor which is paramterized by a single parameter which can be passed as parameter `redshift` (or, equivalently, as `velocity_source`) to the task.
  - Using a parameter `frame` you can specify the reference frame these redshift factors refer to when transforming the spectra to the rest frame of the source. Possible values are 'GEOCENTRIC', 'SSB' or 'LSR'. When starting with the spectra in the rest frame of the spacecraft and applying the task in the form

    ```
    doVelocityCorrection(htp=htp, aux=obs.auxiliary, frame="SSB", -)
    ```

    the spectra are first multiplied by relativistic factors for the transformation from HSO to SSB and then by non-relativistic redshift factors.
  - Using a parameter `redshiftType` you can specify the redshift formula to express how the velocity parameter (`v`)is translated into a redshift factor.
    - `redshift_type="optical"`: `f_source(v) = 1+v/c`
    - `redshift_type="radio"`: `f_source(v) = 1/(1-v/c)`
    - `redshift_type="redshift"` (v='z'): `f_source(v) = 1+v`
    - `redshift_type="relativistic"`: `f_source(v) = sqrt((1+v/c)/(1-v/c))`

    If you specify

    ```
    doVelocityCorrection(htp=htp, aux=obs.auxiliary, frame="SSB",
    redshift_type="optical")
    ```

    the spectra are first multiplied by relativistic factors for the transformation from HSO to SSB and then by a non-relativistic 'optical' redshift factor.

  Alternatively, the parameters are looked up from the meta data of teh timeline product as `frame`, `v_source_frame` and `redshiftType`.

- Algorithms: see Mathematics
- Calibration inputs:
  - The velocity data in some form (see above).

- **Result:**

The frequencies are transformed for all the science datasets included in the timeline product.

- Meta data: The following items are added to the meta data of both, the timeline product and the datasets included therein.
  - 'freqFrame': The rest frame the frequency scale is expressed in.
  - 'redshiftType': the formula used to compute the redshift factorgiven a redshift parameter;
  - 'redshift': the redshift parameter;
  - 'v_frame': a velocity computed from the redshift factor using the 'optical' formula expressed in 'frame'.
  - 'frame': The frame the frequencies are expressed in after executing the task (by default 'LSR' for non-SSO's).

  Note that except for the first ('freqFrame') all the meta data items listed above are added to the timeline product or the datasets only if the target frame has been set to 'source'.
- Datasets: All science datasets found in the timeline product (meta data `'sds_type'='science'`) are transformed.
  - The frequency columns with the name(s) `wavename` or `wavename_i` are transformed i.e. overwritten.
  - The 'LoFrequency' is also Doppler corrected when applying this task. In order to preserve the measured LoFrequency, the latter is copied to a new column called 'LoFrequency_measured'. Note that the comparison of LoFrequency and LoFrequency_measured allows to always return to the original HSO frame. Transforming back to the HSO frame can be achieved by setting the parameter `reverse=True`. Here, the `isVelocityCorrected`-meta data item is set to `False` and `freqFrame` to 'HSO'.
- Calibration output: None.

- **Errors and Warnings:**
  - Warning if the `targetFrame` specified could not be identified - defaults are used in that case.
  - No correction is carried through in case there is already a `freqFrame` meta data item that however cannot be identified.
  - No correction is carried through in case that the `targetFrame` has been set to 'source', but no redshift parameter has been found (as task parameter or as meta data).

    Remark: A redshift parameter set to zero is treated as if no redshift parameter is found.
  - Warning if no velocity information has been found in the data provided to the task. No velocity correction carried through in that case. Quality flag is raised.
  - Warning if not sufficient velocity information has been found in the data provided to the task. Some datasets may be corrected, some not. In the metadata of the timeline product the meta data items are not set.
- **Remarks:**
  - ...
- **Questions, Issues:**
  - ...

5.2.11. DoRadialVelocity

- **Purpose:**

  Corrects the frequency scale for the radial velocity of the spacecraft and possibly of the source by using a non-relativistic approach. Possible target rest frames to transform to (see parameter targetFrame) are "HSO" (short for Herschel Space Observatory), "LSR" or "SOURCE" (for the rest frame of the source). By default, task transforms to the "LSR" frame for non SSO's and "SOURCE" for SSO's.

  Adopting a non-relativistic approach only the radial velocity is needed, i.e. only the component in direction of the pointing of the telescopse. This can be obtained from different sources:

  - from an object of type herschel.share.fltdyn.ephem.Ephemerides passed as 'ephem' parameter to the task:

    ```
    doRadialVelocity(htp=htp, ephem=ephemerides)
    ```

  - from an auxiliary context passed as 'aux' parameter to the task:

    ```
    doRadialVelocity(htp=htp, aux=obs.aux)
    ```

  - from the datasets to be processed, in a column named 'velocity';

    ```
    doRadialVelocity(htp=htp)
    ```

  - from the task parameter velocity_source to specify the velocity of the source relative to LSR, constant in time:

    ```
    doRadialVelocity(htp=htp, aux=obs.aux, velocity_source=10.0)
    ```

    Note that here the source velocity should be expressed in km/sec.

  By default, a positive value entered as velocity_source parameter means that the source and S/C depart from each other (redshift) - note that this has been changed from UR 2.0 to UR 3.0. In contrast, a positive velocity found in the 'velocity' column (which is the radial velocity of the S/C w.r.t. LSR) has just the opposite effect.

- **Description:**

  - Assumptions:

    - ...

  - Mathematics:

    - The frequencies in the spectra are multiplied by the factors

      ```
      1 -- v -/ c
      ```

      where `c` is the speed of light and `v` the velocity of the spacecraft w.r.t. to LSR (non-SSO) or the source (SSO). As a result, a meta data item is added (to both, the datasets and the timeline product) that indicates that the velocity transformation has been applied ('isVelocityCorrected'=True). Furthermore, a meta data item `freqFrame` is set to 'LSR' or 'SOURCE', respectively.

    - The 'LoFrequency' is also Doppler corrected when applying this task. In order to preserve the measured LoFrequency, the latter is copied to a new column called 'LoFrequency_measured'. Note that the comparison of LoFrequency and LoFrequency_measured allows to always return to the original HSO frame. Transforming back to the HSO frame can be achieved by setting

the parameter `reverse=True`. Here, the `isVelocityCorrected`-meta data item is set to `False` and `freqFrame` to 'HSO'.

- In order to transform to the SOURCE rest frame for non SSO's you need to specify a source velocity `velocity_source`code> and set `targetFrame='SOURCE'`. This source velocity is included in the meta data as `v_source_frame`. For consistency with the `doVelocityCorrection`-task the meta data item `frame` which is the frame the source velocity is expressed in is set to 'LSR'.

  You cannot repeatedly correct for different source velocities (as of UR 3.0). In order to try another source velocity do the reverse correction first and appply the modified correction thereafter. Note that in contrast to earlier versions (before UR 3.0) it is no longer sufficient to just set the velocity task parameter to have the spectra to the rest frame of the source - you also need to set the targetFrame parameter.

- Algorithms: see Mathematics
- Calibration inputs:
  - The velocity data in some form (see above).

- **Result:**

  The frequencies are transformed for all the science datasets included in the timeline product.

  - Meta data:
    - 'isVelocityCorrected': Indicates that the correction with the velocities found in the 'velocity'-column has been applied.
    - 'freqFrame': The rest frame the frequency scale is expressed in.
    - 'v_source_frame': The source velocity used in the velocity correction.
    - 'frame': The frame the source velocity is expressed in (always LSR in this task).
  - Datasets: All science datasets found in the timeline product (meta data `'sds_type'='science'`) are transformed.
  - Columns: The frequency columns with the name(s) `wavename` or `wavename_i` are transformed i.e. overwritten.
  - Calibration output: None.

- **Errors and Warnings:**
  - No velocity information has been found in the data provided to the task.

- **Remarks:**
  - ...

- **Questions, Issues:**
  - ...

5.3. Level 2 Pipeline

5.3.1. DoCleanUp

- **Purpose:**

  Remove the data from the timeline product which is no longer used in the level2 processing or in the end user analysis. To be specific, all datasets that are of not of type 'science' (`sds_type`) or which are not of type 'science' and correspond to 'ON' measurements are removed. Furthermore, the science datasets that belong to the same LO tuning group and/or the same raster point and/or the same scan line number are merged to form new datasets. Datasets are merged only up to a configurable maximum number of scans.

```
doCleanUp(htp = htp, params = params)
```

```
doCleanUp(htp = htp, retain='science', mergeDatasets=True, datasetSize=50)
```

- **Description:**

  - Assumptions:

    - The datasets included in the timeline product have a meta data field 'sds_type'.

    - The `CheckFreqGrid` has been processed so that the the meta data item `frequencyGroup` is available in the datasets.

    - The datasets belonging to the same group have consistent segmnet shapes so that merging is possible.

  - Mathematics:

    - With the parameter `retain` you can tell the task what data types you would like to keep. Possible values are

      - 'scienceOn'

      - 'science'

    - Merging of the datasets: The datasets that

      - belong to the same LO tuning group,

      - correspond to the same raster point in raster maps (the same values in the 'rasterRowNum'-'rasterColumnNum'-columns), and

      - correspond to the same scan line in otf maps (the same value in the 'scanLineNum'-column) are merged (ON and OFF datasets are merged separately). In this way, larger datasets and fewer products are included in the result timeline product. With the parameter `mergeDatasets` you can specify whether to merge at all or not (`True` or `False`). The maximum size of the resulting datasets can be specified by the parameter `datasetSize`. In this merging of the datasets the scans are not split so that the scans of a given original dataset always end up in the same result dataset.

    - The summary table of the timeline product is updated.

    - Setting the parameter `ignore=True` the execution of the task can be omitted.

  - Algorithms: see above.

  - Calibration inputs: none.

- **Result:**

- Meta data: Nothing added.
- Datasets: Merged datasets as described above.
- Flags: None.
- Errors and Warnings:
  - None

- **Remarks:**
  - ...
- **Questions, Issues:**
  - ...

5.3.2. DoAntennaTemp

- **Purpose:**

  Correct for all telescope dependent parameters except the coupling of the antenna to the source brightness distribution, i.e. translate to a `T^*_A` (`T_A`-star) scale where `T^*_A = T'_A/eta_l` with forward efficiency `eta_l`.

```
doAntennaTemp(htp = htp, cal = forwardEffTable)
```

```
doAntennaTemp(htp = htp, forwardEff = 0.68)
```

  For extended sources typically rather the `DoMainBeamTemperature` is applied.

- **Description:**
  - Assumptions:
    - The correction can be applied to all the spectra found in the timeline product. Therefore, the `DoCleanUp`-task should have been processed before so that only 'science'-datasets are included in the timeline product.
  - Mathematics: All the 'flux' data found in the spectra (at this time typically already intensities) are multiplied by a scalar factor which is given by the inverse of the forward efficiency. The forward efficiency is obtained either
    - by passing a custom forward efficiency by setting the `forwardEff`-parameter to a suitable value or
    - by passing the reference to a calibration product of type `GenericPipelineCalProduct` that contains tables with the forward efficiencies. Here, the task parameter `cal` needs to be set accordingly.
  - Algorithms: see above.
  - Calibration Input: The forward efficiencies obtained from the calibration tree. Note that these typically can be retrieved from the calibration tree or from within the observation context by

```
forwardEffTable =
obs.calibration.getCalNode("downlink").getCalNode("generic").getProduct("forwardEfficiency-
H")
```

for H-polarization and similarly for V-polarization. Note that the tables included in these products should be of type `GenericPipelineCalTable` and should look as shown in the following Figure.

Figure 5.6. Sample table with forward efficiencies

- **Result:**

  All the flux column entries in the science datasets are re-scaled.

  - Meta data: None
  - Datasets: All flux column entries in the science dataset are rescaled. No other other is modified.
  - Flags: None
  - Errors and Warnings: None

- **Remarks:**

  - ...

- **Questions and Issues:**

- • ...

5.3.3. MkSidebandGain

- **Purpose:**

  Compose a class that provides the sideband gains coefficients dependent on detector band, sideband, LOF-scale and IF-scale. These coefficients will subsequently be applied in the `DoSidebandGain`-task. Currently, the task returns an instance of `herschel.hifi.pipeline.generic.cal.CalSidebandCoeffImpl` (which implements the interface `herschel.hifi.pipeline.generic.CalSidebandCoeff`). This implementation only allows for 'orthogonal' dependencies on LOF and IF in the sense that the relative changes of the coefficients along the IF scale are independent of the LO frequency.

```
                gains = mkSidebandGain(htp=htp, shape=sidebandGainIF,
level=sidebandGainLO)
```

  where `sidebandGainIF` and `sidebandGainLO` are calibration products that contain tables with the IF- or the LOF- dependency, respectively. These products (of type `GenericPipelineCalProduct`) typically are obtained from the calibration tree. In case you want to work just with default coefficients 0.5 you can skip this task and call `DoSidebandGain` without passing a `cal` task parameter.

- **Description:**

  - Assumptions:
  - Mathematics:

- A calibration products with tables specifying the LO dependency and another calibration product specifying the IF dependency are passed to the task with the parameters `level` and `shape`, respectively. Both products should be of type `GenericPipelineCalProduct`. The task does nothing more than extracting from these products suitable tables corresponding to the detector band and the start date of the observation. Note that for different periods of the life time of the instruments different tables could be applicable due to instrument drifts.

- After extracting these tables an object of type `CalSidebandCoeffImpl` is composed. This object provides access to the channel-specific sideband gains coefficients by the method `getSidebandGainCoefficients(double lof, Double1d ifFreqOut, boolean usb).` The coefficients are composed by

  - extracting interpolated values from the table with the IF dependency - using the `ifFreqOut`-array. The output is of the same length as `ifFreqOut`.

  - Next, from the table with the LO dependency as suitable level is interpolated (passing the LO frequency of the given scan). This defines a factor the array obtained in the previous steps is multiplied with.

  For the interpolation used to retrieve suitable values from the tables, a linear scheme is applied.

- Algorithms: see Mathematics.

- Calibration Input:
  As mentioned above, two calibration products are passed here - one with the IF- and another one with LOF-dependency. The tables included in these products should have the columns as shown in the sample tables below.

Figure 5.7. Sample table specifying the LO dependent gain levels.

Figure 5.8. Sample table specifying the IF dependent gain shape.

Note that different grid points are specified in different lines of the table (see the table with the IF dependency). The calibration products can be retrieved from the calibration tree in the observation context by

```
sidebandGainLO =
obs.calibration.getCalNode("downlink").getCalNode("generic").getProduct("sidebandGainLO-
H")
```

```
sidebandGainIF =
obs.calibration.getCalNode("downlink").getCalNode("generic").getProduct("sidebandGainIF-
H")
```

for the H-polarization and similarly, for the V-polarization.

- **Result:**

No changes on the original timeline product product.

- No changes in the input timeline product.

- Calibration Output: Implementation of the interface `CalSidebandCoeff`. Note that this is not a product hence cannot be persisted.

- Flags: No flags.

- Errors and Warnings: Warning if no IF-shape or LO-level table could be obtained (either from within the associated calibration products or since no such calibration products have been passed to the task.

- **Remarks:**
  - ...
- **Questions, Issues:**

5.3.4. DoSidebandGain

- **Purpose:**

  Divide the flux (at this stage typically an intensity) by the sideband-specific, detector-band specific, LOF- and and IF-dependent gain coefficients.

```
htpUpper = doSidebandGain(htp=htp, cal=gains, sideband="upper")
```

  where gain is an object that provides the gains coefficients as composed by the `MkSidebandGain`. Note that this gains object is required to implement the `CalSidebandCoeff`-interface defined in `herschel.hifi.pipeline.generic`.

  If the task parameter `cal` is not set default coefficients equal to `0.5` are applied.

```
htp = doSidebandGain(htp=htp)
```

  Furthermore, if the `sideband` parameter is not set the default `"both"` is applied. In this case, result of applying the USB coefficients is returned when executing the task and the timeline product with the LSB coefficients is obtained by

```
htpLower = doSidebandGain.image
```

- **Description:**
  - Assumptions:
    - The spectrum data included in the timeline product should be given at the IF frequency scale, i.e. not yet transformed to the sky frequency scale.
  - Mathematics:
    - The gain coefficients are provided to the task in form of a `CalSidebandCoeff`-object which is passed as `cal`-parameter. The task retrieves the LOF- and IF-dependent coefficients by using the method `getSidebandGainCoefficients(double lof, Double1d ifFreqOut, boolean usb)`. The array with the IF frequencies is passed as input so that the `cal`-object can provide interpolated values. Note that here it is important that the spectra are expressed at the IF frequency scale.
    - In general, the gain coefficients depend on the sideband. The task parameter `sideband` indicates what sideband the coefficients should refer to:
      - `sideband='upper'`: The coefficients for the upper sideband are applied and one timeline product is returned as result.
      - `sideband='lower'`: The coefficients for the lower sideband are applied and one timeline product is returned as result.
      - `sideband='both'`: Two timeline products are returned - one with the coefficients for the upper sideband and the other with the coefficents for the lower sideband. The time-

line product with the upper sideband coefficients is returned when calling the task (either `htpUpper=doSidebandGain(htp=htp, cal=gains, sideband="both")` or `doSidebandGain(htp=htp, cal=gains, sideband="both")` and `htpUpper=doSidebandGain.htp`). The timeline product with the lower sideband coefficients is obtained as `htpLower=doSidebandGain.image`.

- The task modifies all the fluxes (or intensities) found in all the science data (see meta data field `sds_type`) of the timeline product by dividing the 'flux' arrays by the array with the coefficients.

- Algorithms: Nothing to add here.
- Calibration inputs:
  - A calibration object of type `CalSidebandCoeff` which is an interface-type. Note that this allows to pass different implementations of this interface as calibration input - such as implementations that provide the coefficients with a more general dependency on the LOF and IF.

- **Result:**

  All the 'science' datasets are re-scaled (see the meta data field `'sds_type'`) of the datasets included in the timeline product) - all other datasets remain un-touched.

  - Meta data: A field with name `sideband` and values `'USB'` or `'LSB'`, respectively is set for the timeline product and all science datasets inlcuded therein.
  - Datasets: All datasets with a meta data field `sds_type` set to `'science'` are modified.
  - Columns: Columns with name 'flux' or 'flux_i' where i indexes the segment number.
  - Calibration Output: No calibration output.

- **Errors and Warnings:**
  - ...

- **Remarks:**
  - In case no or no valid gains calibration coefficients can be retrieved default coefficients (0.5) are applied.

- **Questions, Issues:**

5.3.5. ConvertFrequencyTask

- **Purpose:**

  In this step of the pipeline, the spectra are transformed from the IF frequency scale to the sideband frequencies. For detector bands 1-5 this is defined by

  ```
  f_usb = f_LO + f_IF  and f_lsb = f_LO -- f_IF ,
  ```

  respectively. For the bands 6 and 7,

  ```
  f_usb = f_LO + CF -- f_IF  and f_lsb = f_LO -- CF + f_IF ,
  ```

  where the conversion factor CF is given by 10.4047 GHz for horizontal and 10.4032 GHz for vertical polarization. At the same time, the units are changed from MHz to GHz. This task, the `ConvertFrequencyTask`-task included in the package `herschel.hifi.pipeline.product`, is not a dedicated pipeline task - rather it is designed to be used in the interactive analyis as well so that the user can switch forth and back between the different scales. As an example, the task can also be used to transform to the velocity scale.

```
frequencyConverter=ConvertFrequencyTask()
```

```
frequencyConverter(htp=htpUpper, to='usbfrequency')
```

```
frequencyConverter(htp=htpUpper, to='lsbfrequency')
```

The task modifies the timeline product in place - it iterates over all the datasets included in the timeline product and transforms the frequency scale.

Note that once the sideband gain calibration has been carried through, usb data should not be transformed to a lsb frequency scale (and vice versa).

- **Description:**
  - Assumptions:
    - None
  - Mathematics:
    - In the standard pipeline processing transform from IF frequencies (given in MHz) to sideband frequencies (given in GHz) where the sideband frequencies are defined by

      ```
      f_usb = f_LO + f_IF  and f_lsb = f_LO -- f_IF
      ```

    - When transforming to the velocity scale, a reference frequency corresponding to zero velocity is specified. To give an example:

      ```
      frequencyConverter(htp=htp, to='velocity', reference=1902.055688)
      ```

      For this to work, the reference frequency should be expressed in GHz. Then, the standard Doppler-shift formula is used to transform to the velocity-scale:

      ```
      v = -- (f -- f_ref) -/ f_ref * c
      ```

      where `c` is the speed of light expressed in units km/sec.

    - The result of the transformation is written to new columns, named as `wavename` or `wavename_i` where `wavename` is the name of the new frequency scale ('usbfrequency', 'lsbfrequency', 'velocity' or 'frequency') which is also set in the `wavename` meta data field.
  - Algorithms: see Mathematics
  - Calibration inputs:
    - None
- **Result:**

  The frequencies are transformed for all the datasets included in the timeline product.

  - Meta data:
    - The 'wavename' meta data field is changed and set to 'usbfrequency', 'lsbfrequency', 'frequency' or 'velocity'.
    - The 'sideband' meta data field in the datasets is set to 'usb' or 'lsb'.
    - When transforming to a velocity scale, a meta data item named 'referenceFrequency' is set to the 'reference' (frequency) specified in the task parameter.

- Datasets: All datasets found in he timeline product are transformed.
- Columns: The frequency columns with the name(s) `wavename` or `wavename_i` are transformed where `wavename` is the name of the frequency scale prior to the transformation (see the `getWaveName()`-method of the datasets). The result of the transformation is written to new columns (named `wavename` or `wavename_i` with the new wavename ('usbfrequency', 'lsbfrequency', 'frequency' or 'velocity', respectively).
- Calibration output: None.

- **Errors and Warnings:**
  - ...
- **Remarks:**
  - ...
- **Questions, Issues:**
  - ...

5.3.6. MkFreqGrid

- **Purpose:**

  Creates a linear frequency grid that can be used `DoFreqGrid`-task to resample the spectra to. By default, the width between successive gridpoints is set to 0.5 MHz for WBS data - for HRS data it is determined by inspecting the input spectra. The algorithm described below is applied on a per LO tuning group basis. The algorithm is designed such that for WBS data the grids for upper sideband data and lower sideband data and for the different LO tuning groups have one common underlying linear grid.

```
outputGrid = mkFreqGrid(htp=htp)
```

```
outputGrid = mkFreqGrid(htp=htp, stepsize=0.5, unit="MHz")
```

- **Description:**
  - Assumptions:
  - Mathematics:
    - The equidistant frequency grid(s) are determined on a per LO tuning group basis. Actually, within each group, one grid is constructed for each subband / segment. Herefore, the following procedure is adopted:
      - Assume the array with the frequencies vector is called `f` and the stepsize is denoted by `s`. First, the minimum and the maximum frequency is determined for each subband by looping over all scans within a LO tuning group. We denote them by `minF` and `maxF`, respectively.
      - These minimum and maximum frequencies are rounded to the nearest integer multiple of the stepsize. From these rounded minimum and maximum frequencies we compute the number of grid points (n) in the output grid. Note that this step guarantees that all WBS spectra have one common underlying linear grid.

- For frequencies in the LSB, the frequencies are first mirrored to the upper sideband so that the rounding procedure leads to identically sized output grids. For the mirroring, we also check whether a velocity correction has been applied in which case the LO frequency is adjusted accordingly (see the meta data fields 'isVelocityCorrected' or 'velocity_user' and the `DoRadialVelocity`-task).

- The grid of size `n` is then created starting for the IF or USB (or LSB) with the rounded minimum (or maximum) frequency increasing (or decreasing) in steps of stepsize.

- In case no stepsize is specified, it is set to 0.5 for WBS and for HRS the 'channelSpacing' meta data value which can be found in the datasets of the timeline product is used. If no such field in any of the datasets can be found, a stepsize is computed with the following rule: From the `minF` and the `maxF` determined for the frequency group and the number of channels (`n`), the stepsize is set to

```
stepsize = 1/max(1,round(1/s)) where s =(maxF-minF)/(n-1)
```

  when all involved quantities are expressed in MHz.

- Algorithms: See Mathematics.

- Calibration input: None

- **Result:**
  The original timeline product is not modified by this task.

  - Meta data: No changes.

  - Columns: No changes.

  - Calibration output:
    The result of this task is a product of type `CalFreqGrid` in which the grids per LO tuning can be found. For each LO tuning, a separate table dataset contains the frequency grids per subband. Furthermore, these frequency arrays are included in the table under the same name and unit as in the original timeline product.

Figure 5.9. Calibration product containing the frequency grids created by the MkFreqGrid task.

In the figure above you can see a sample output of the `MkFreqGrid`-task. For each LO tuning group with group id 'k' a table dataset with key 'group_k' is included in the product.

- **Errors and Warnings:**

  - Warning if the wave scale unit could not be identified from the data. In this case it is assumed that the unit of the stepsize is expressed in the same units as the wave scale the spectra are expressed in.

- Remarks:

  - ...

- Questions, Issues:

  - ...

5.3.7. DoFreqGrid

- **Purpose:**

Resamples the spectra to the frequency grid specified as input task parameter. Various options are available:

- Specify the grid as the output of the `MkFreqGrid`-task, i.e. a product of type `CalFreqGrid`.

```
doFreqGrid(htp=htp, grid=grid)
```

- Specify the grid as a PyDictionary with the dataset key in the timeline product as key and with an array of grids, one for each subband, as values (`Double1d[]`).

```
doFreqGrid(htp=htp, grid = {1:grid1, 2:grid2}) where grid1 and grid2 are
suitable Double1d[].
```

- Specify the grid by just a resolution parameter:

```
doFreqGrid(htp=htp, resolution=0.5, unit="MHz")
```

Here, per LO tuning and subband an output grid is constructed with stepsize given by half the resolution parameter and by starting with the minimum frequency found per subband and LO tuning group.

- **Description:**
  - Assumptions:
    - ...
  - Mathematics:
    - For the actual resampling the functionality available in the spectrum toolbox is used (see `herschel.ia.toolbox.spectrum.ResampleFrequency`). For the documentation of the resampling scheme we refer to the spectrum toolbox documentation.
    - Currently, the resampling scheme is not configurable. The resampling scheme set as default in the resampling task of the spectrum toolbox is applied. This is set as a trapezoidal integration scheme in combination with a linear interpolation scheme. The integration scheme is needed to assure that the resampling scheme preserves the integrated intensity (up to within the order of the scheme).
    - In addition to the flux and the wave data, the flags and the weights are also modified by this operation. The flags of all the channels involved in the computation of an output grid point are propagated using a bit-wise OR. The weights are processed using the same resampling scheme as for the flux.
    - The grids the spectra should be resampled not necessarily need to be linear (equidistant). This could be used to do the co-add: Select the frequency arrays from one scan for each dataset e.g. of WBS-H. Put that all in a PyDictionary. Pass this to dictionary together with the WBS-V data to this task. Thereafter, you can loop over the datasets included in the timeline product and do a pairwise average which is identical with the add (+).
  - Algorithms:

    The input spectrum provides an array of frequencies whose values are interpreted as the mid-points of flux-channels. The flux array is assumed to give either the flux density for each of these channels (`density=True`, i.e. the flux per frequency unit) or the integrated flux seen in the channel (`density=False`). The channel boundaries are constructed by the mid-points of the frequency values - so, strictly speaking, if you have an irregular grid, the frequency values are not mid points within the channels. Similarly, from the new frequency grid - the user wants to resample to - suitable frequency channels are defined.

Now, there are different schemes available for how the flux values in the output grid are defined. The scheme toe be used in the pipeline task can be configured by using the task parameter `scheme`. By default (or without setting the `scheme` parameter, the 'euler' scheme is used in the pipeline. The following scheme are available:

- 'euler': All the flux contributions defined for the original grid are summed up - for not completely overlapping channels, the fractional contribution are taken where the fraction is determined just by comparing the width of the overlapping part of the new with the channel width of the orginal spectrum.
- 'trapezoidal': Here, the flux density function defined by the input grid is integrated by using a trapezoidal integration scheme. At the boundaries of the new channels, the flux density values are computed from the original grid by using a linear interpolation scheme.
- 'normal': not yet available - planned for UR 5.0.

- Calibration input: Calibration product with the grids the spectra should be resampled to.

- **Result:**
  - Meta data:
    - Resampling width use in the resampling algorithm (`frequencyWidth`).
    - New resolution of the spectra is added to the meta data, called `resolution_resampled`. It is defined approximately as `sqrt(r0*r0 + w*w)` where `r0` is the original resolution and `w` the resampling width.
  - Spectral segment data changed - as described in the Mathematics. All the other columns remain unchanged.
- **Errors and Warnings:**
  - ...
- **Remarks:**
  - ...
- **Questions, Issues:**
  - Task should allow to configure the resample scheme - though the Gaussian resampling scheme is not yet available.
  - ...

5.3.8. DoAverage

- **Purpose:**

  Compute the average over different scans that belong to the same LO tuning group (frequency surveys), the same raster columna and row (in raster maps) or the same line number in OTF maps. Furthermore, science data from ON or OFF are not mixed. Various different options for how to do the average and for pre-selecting the scans to be averaged are available. With the `return_single_ds` you can specify whether you would like to have the result as a dataset or included in the original timeline product. The standard pipeline mode, you see the following form:

```
htp = doAvg(htp=htp, return_single_ds=False)
```

  or

```
htp = doAvg(htp=htp, params=params)
```

where `params` is a `PipelineConfiguration`-object e.g. with the information that the `return_single_ds`-parameter set to False should be used. With the parameter `selection_meta` you can specify a list of types, specified in the the meta data item 'sds_type', the averaging should be restricted on:

```
ds = doAvg(htp=htp, selection_meta=["science", -"hc"], return_single_ds=True)
```

The values "scienceOn" and "scienceOff" are also possible values that can be entered in this list (although these are not values that are found in the 'sds_type' meta data field:

```
ds = doAvg(htp=htp, selection_meta=["scienceOff"], return_single_ds=True)
```

More generally, you can formulate restrictions on other string-valued meta data items by using py dictionaries:

```
ds = doAvg(htp=htp, selection_meta={"frequencyGroup":["1","2"]},
return_single_ds=True)
```

Even more generally, you can restrict the selection to be averaged on scans for which some other columns match given values:

```
ds = doAvg(htp=htp, selection={"bbtype":[6005,6031]}, return_single_ds=True)
```

Here, only the scans that have a value 6005 or 6031 in the column 'bbtype' of any of the datasets included in the timeline product.

- **Description:**
  - Assumptions:
    - Input data is a timeline product.
    - All datasets that belong to the same group to be averaged (see below) are assumed to have the same subband shapes.
    - The spectra that belong to the same group to be averaged are assumed to have frequency scales close enough so that frequency resampling can be ignored. If not, apply `doFreqGrid` first.
  - Mathematics:
    The average task delegates most of the actual averaging to the average task in the spectrum toolbox `avg` (`herschel.ia.toolbox.spectrum.AverageSpectrum`).
    - In a first step the timline product is analyzed for what scans should be considered for the average. Dedicated task parameters are available to specify this selection:
      - `selection_meta`: Here you can formulate conditions on the meta data fields for the datasets to be selected. Possible values are
        - Python-List: With a python list with string values you can specify the types of datasets the average(s) should be restricted on. This type information is found in the meta data field 'sds_type' of the datasets. Typical values are `science`, `hc`, `comb`. Further possible values that are not values of the 'sds_type'-field are `scienceOn` and `scienceOff`. These can be used to restrict to science data from the On or the Off position, respectively.
        - `Python-Dictionary`: With a python dictionary with meta data keys as keys and the Python lists as values. With the Python list you specify the admissible values for the meta data field(s) so that a given dataset is included in the average. In case more than one meta data key is specified, the different conditions are combined with a logical AND. Example: `selection_meta={"frequencyGroup":["1","2"], "sds_type":`

`["science","hc"]}` - here, only datasets with the meta data field 'frequencyGroup' set to "1" OR "2" AND the 'sds_type' set to "science" OR "hc" are selected.

- `selection`: Here you can specify conditions on the table data of the datasets so that specific scans ("rows") are selected. Possible values are

  - Python List: Specify the row numbers to be selected from each dataset, possibly preselected with the meta data filter. This option may seem not very useful but is inherited from the toolbox task.

  - Python Dictionary: Specify lookup criteria with keys specifying the name of the column to formulate the condition with and values specifying a list of admissible values that should be matched so that a given row is selected. In case more than one key-value pair is specified, the different conditions are combined with a logical AND. Example: `selection={"bbtype":[6031,6032], "buffer":[1]}` - here, only scans with the bbtype equal to 6031 OR 6032 AND the buffer equal to 1 are selected.

  - General selection model (interface of type `SelectionModel`: See the documentation of the spectrum toolbox or in the user reference manual (e.g. `AverageSpectrum`)for further details.

  In case this selection functionality should not yet be sufficient, make a pre-selection by running the task `herschel.hifi.pipeline.util.tools.SelectSpectrum` before applying the `doAvg`. See the HIFI user reference manual for further details on that task.

- In a next step, suitable groups are formed with the spectra that have been selected for the average. This step can be skipped by setting the `preserveGroups` task parameter set to `False`. Scans belonging to different groups are not combined in the average. This means that for each group one average is computed. The following criteria are used to form the groups:

  - Type: Datasets with different types specified in the `sds_type` meta data field are not combined in the average.

  - ON / OFF: Spectra from the ON and the OFF position are not mixed.

  - LO Tuning Group: Scans with different freuqnecy groups specified with the `frequencyGroup` meta data field are not combined (see `CheckFreqGrid`).

  - Raster Point: Scans associated with different raster positions in raster maps are not mixed. Here, the meta data fields `rasterColumnNum` and `rasterLineNum` are used. These fields are added in the `DoPointing`-task to the timeline product.

  - "Line scans": Scans associated with different lines in OTF maps are not combined in the average. Here, the meta data field `scanLineNum` is used which is also added in the `DoPointing`-task to the timeline product.

  Note that the creation of the timeline product and the pipeline pipeline processing assure that scans within the same dataset cannot belong to different groups. In particular, this is checked and assured by running the task `checkDataStructure` in which datasets with different `bbnumber` are split.

- Once the groups are built, the averages can be computed on a per group level. Here, the functionality of the spectrum toolbox is used. Accordingly, with the parameter `variant` you can specify whether weights and flags should be included in the computation of the average:

  - `variant="flux"`: arithmetic average of the flux, frequency scale set to frequency scale of the first scan found for the group, weights are added, flags are propagated with bitwise OR.

  - `variant="flux-weight"`: weighted average of the flux, frequency scale set to frequency scale of the first scan found for the group, weights are added, flags are propagated with bitwise OR.

- `variant="flux-flag"`: arithmetic average of the flux where for a given channel only unflagged values are included in the average - unless for a given channel only flagged channels are available, the frequency scale is set to the frequency scale of the first scan found for the group, weights are added, flags are set to zero or propagated with bitwise OR, respectively.

- `variant="flux-weight-flag"`: The combination of the previous two alternatives.

In addition to this configuration for the actual spectrum data, the average operation is also configured for the other 'attributes' (columns) of the datasets.

- Average: `obs time, Chopper, cmd_chopper, LoFrequency, frequency_monitor, hot_cold, MJC_Ver, MJC_Hor, longitude, latitude, velocity, posAngle, longitudeError, latitudeError, velocityError, posAngleError, tsys_median`.
- Add: `scancount, integrations, integration time`.
- Unique (include the unique value found in case all scans have the same unique value): `bbtype`.
- Max: `packet time`.
- (Bitwise) OR: `row flag`.
- AND: `frmon_valid`.

- Finally, the result is either returned as a `HifiSpectrumDataset` if the parameter `return_single_ds` has been set to `True` (default) or the individual averages are included in the original timeline product while all the original datasets are removed. In the first case, each line in the result will correspond to one of the group averages. You can identify the groups from the bbtype, LO Frequency, rasterLineNum/rasterColumnNum, or scanLineNum. Note that, typically, comb datsets cannot be included in the average since these have different segment shapes.

- Algorithms: See Mathematics.
- Calibration input: None.

- **Result:**
  - Meta data: For each group, the meta data of the first dataset assigned to the group is copied.
  - Spectral segment data: Depending on the task parameter `variant` the flux, weight, flag and wave data are processed differently (see Mathematics for further details).
  - Columns: see Mathematics for further details on how the data found in other columns are processed.
- **Errors and Warnings:**
  - In case a single dataset belongs to different groups at the same time no averageing is done and a warning is given that the `checkDataStructure`-task should be run first.
  - In case one of the group averages could not be completed (e.g. not all scans could be considered for the average so that the average is incomplete) a warning is created.
- **Remarks:**
  - ...
- **Questions, Issues:**

- For cross maps, data belonging to different points are mixed in the current settings.
- ...

5.3.9. DoFold

- **Purpose:**

  Perform the folding for frequency switched spectra.

```
doFold(htp=htp)
```

```
doFold(htp=htp, throw = --60, unit="MHz")
```

  The folded spectra are constructed by averaging the original spectra with a shifted and inverted copy - shifted by the LO throw. This algorithm corresponds to the most simple scheme described in the article *"Recovering line profiles from frequency-switched spectra", H.Liszt, Astron. Astrophys. Suppl. Ser. 124, 183-188 (1997)*. For the input spectra included in the input timeline product a linear frequency scale is assumed. The size of the spectra is reduced by the number of channels corresponding to the LO throw.

- **Description:**
  - Assumptions:
    - The frequency grid of the input spectra should be linear (equidistant).
  - Mathematics:
    - The original spectra are averaged with a copy which is shifted by the LO frequnecy throw and inverted.

      Note that the folding is not perfect in the sense that typically, ghost lines appear to the left and the right or the actual lines as asorption lines. This can be annotated with the sequence -,-'-,-.

      Some care is needed when interpreting e.g. USB data containing both USB and LSB spectra: USB emission lines appear as emission lines and LSB lines appear as "absorption lines". This is explained in some more detail in the following:

      - An emmission USB line will show up in the USB as -,-'- if the LO throw is positive (i.e. the reference phase with the line pointing downwards is shifted by the LO throw to the left). Applying the fold to this timeline product (corresponding to the USB) leads to a picture of the form -,-'-,- where a duplicated reference phase occurs phase to the right of the emission line. In case the LO throw is negative the reference in the input spectra will appear on the right and the duplicated ghost in the folded spectra appears on the left.

      - An LSB emission line will show up in the USB as -'-,- if the LO throw is positive (here the reference phase with the line pointing downwards is shifted by the LO throw to the right). Applying the fold to this (USB) timeline product leads to a picture of the form -'-,-'- where a duplicated 'emission' line appears to the right.

    Below, an example is shown for an emission line in the LSB before (blue) and after (red) the fold-operation. Note that the line appears as emission line on the LSB frequency scale, but as asorption line on the USB frequency scale.

Figure 5.10. An LSB emission line plotted on the LSB frequency scale.

Figure 5.11. An LSB emission line plotted on the USB frequency scale.

  - The LO throw parameter can either be passed as parameter `throw` or it is retrieved from the meta data of the timeline product using the key `loThrow`. Note that this information is added to the timeline product in the `checkFreqGrid`-task. You can specify the unit of the throw specified in the task using the parameter `unit` with typical values `'MHz'` and `'GHz'`. If no unit is specified the throw specified in the task signature is assumed to have the same unit as the freuency of the spectra in the timeline product.

  - Setting the `shift` to `True` shifts the folded spectra by half the throw in direction of the throw.

- Algorithms: See Mathematics.

- Calibration input: None.

- **Result:**

  - Meta data: None

  - Spectral segment data: Changes just the frequency scale. No other changes.

- **Errors and Warnings:**

  - Warning is given and no data processed when no `throw` has been specified and no `loThrow` is found in the meta data of the timeline product.

- **Remarks:**

  - ...

- **Questions, Issues:**

  - ...

5.3.10. DoSpurs

- Purpose: Detect and remove spurious signals in the scans.

- Description:

- Result:

- Remarks:

- Questions, Issues:

5.3.11. DoStitch

- **Purpose:**

  Task for stitching the subbands of the scans included in a HifiTimelineProduct. The stitching is performed on a per point spectrum (scan) basis. The result is again a point spectrum that still may consist of several segments - in case gaps are found between consecutive segments. The way how the stitching is done is defined by the parameter 'variant' (see below). The task replaces the datasets in the timeline product by new datasets with the stitched spectra. The actual stitching is performed per dataset by calling the `StitchSpectrum`-task in the spectrum toolbox (see `herschel.ia.toolbox.spectrum`. Since the shape of these spectra may be different after

stitching the spectra need to be resampled in the general case. Here, the sampling width can be specified using the stepsize parameter. In case no or a zero stepsize is specified, resampling is avoided if all the stitched point spectra have the same shape. With the `selection_meta`-parameter a restriction can beformulated on which spectra should be stitched. Note that the spectra no stitched remain unchanged in the output timeline product.

```
 htp = doStitch(htp=htp)
```

```
 htp = doStitch(htp=htp, variant="crossoverPoints", edgeTolerance=0.1,
stepsize=1.0, unit="MHz")
```

```
 htp = doStitch(htp=htp, variant="midPoints", stepsize=1.0, unit="MHz")
```

```
 htp = doStitch(htp=htp, variant="splitPoints", splitPoints = [5000.0, 6000.0,
7000.0], stepsize=1.0, unit="MHz")
```

```
 htp = doStitch(htp=htp, variant="average", stepsize=1.0, unit="MHz",
avg_variant="flux")
```

```
 htp = doStitch(htp=htp, selection_meta = ["science", -"hc"])
```

- **Description:**
  - Assumptions:
    - Input a `HifiTimelineProduct` with calibrated frequency scales (i.e. processed through the instrument pipelines up to level 0.5).
  - Mathematics:
    - The actual stitching is processed on a per point spectrum (scan) basis (the rows in the datasets). The stitched spectra that originate from the same dataset are resampled to a common frequency grid so that they can be included in a new dataset. These datasets replace the original datasets in the timeline product. Hence, the task modifies the input timeline product but not the datasets.
    - For each scan, the overlapping ranges are determined. For WBS spectra, typically exactly two segments participate in a given overlap range. For HRS spectra, in principle, more than two 'subbands' can contribute to a overlap range. There are different options of how the overlapping ranges are treated - the options can be called by setting the `variant`-parameter. Generally, we distinguish the algorithms capable of handling only two subbands per overlap range, the "cut and concatenate" algorithms, and the algorithms capable many subbands per overlap range. We first list the "cut and concatenate" options:
      - Cut and stitch at cross over points (`variant="crossoverPoints"`): Here, cross-over points are taken to cut the spectra. These are determined by looking for the minimum distance of the flux values. In case more than one minimal point is found the point closest to the center of the overlap range is taken. In order to avoid selecting points close to the border of the overlap point the parameter `edgeTolerance` can be set e.g. to 0.1 that will restrict the range to search the cross-over points to 80% of the original overlap range.
      - Cut and stitch at mid points: The points to cut the spectra are specified as the mid points of the overlap ranges.
      - Cut and stitch at predefined points: Using the parameter `stitchPoints` you can specify the values at the the spectra should be cut (`stitchPoints = [5000.0, 6000.0, 7000.0]`). The length of the list should be equal to the number of overlap ranges in the spectra.

    Once the points to cut the spectra in the overlap ranges are specified, the segments are glued by simply concatenating them. For the second category, we only have a single option available at the moment:

- Resample and average overlapping parts: Here the spectra are resampled in the overlap range to a common grid. This grid is specified as a linear grid with stepsize given by the task parameter `stepsize`. In case a `stepsize=0` is specified, resampling is omitted if possible (i.e. if the number of grid points found in the overlap range is the same for all the subbands contributing to the overlap range. With the parameter `avg_variant` you can specify the kind of average to be applied - typically:
  - `avg_variant="flux"`: straight average of the flux values
  - `avg_variant="flux-weight"`: weighted average of the flux values
  - `avg_variant="flux-flag"`: straight average of the flux values, ignore flagged values
  - `avg_variant="flux-weight-flag"`: weighted average of the flux values, ignore flagged values
- The stitched spectra originating from the same dataset are resampled to a linear frequency grid with stepsize given by the task parameter `stepsize`. In case a `stepsize=0` is specified, resampling is omitted if possible.

- Calibration input: None.

- **Result:**
  - Meta data: None
  - Spectral segment data: Stotched segments as described in the section 'Mathematics' above.

    Other columns: Copied from the input spectra.
- **Errors and Warnings:**
  - An exception is thrown in case an option is specified which is suited for two subbands per overlap range - in the spectra, however, an overlap range with more than two contributing subbands have been found.
  - Warning if no cross-over point could be identified. In this case the mid points are used instead.
- **Remarks:**
  - ...
- **Questions, Issues:**
  - ...