HIFI User's Reference Manual

Herschel Data Processing

Version 3.0, Document Number: HERSCHEL-HSC-DOC-0935
05 October 2010

HIFI User's Reference Manual: Herschel Data Processing

Table of Contents

Categorized view of Commands

This chapter provides a categorized view of all built-in DPfunctions, tasks and objects.

DP Access

- AccessDataFrameTask - DP Access Module: AccessDataFrameTask
- AccessHkParamTask - DP Access Module: AccessHkParamTask
- AccessPacketTask - DP Access Module: AccessPacketTask

HIFI

- DoGridding - produces cube(s) of images from a HifiTimelineProduct.
- GriddingTask - produces a cube of images from a HifiTimelineProduct,

HIFI Diagnostic Housekeeping

- ViewIVcurveScansTask - Task for plotting IV-curve scan data from array of TM-packets

HIFI pipeline task

- DoHrsBadChans - Step2a: Task which corrects the correlation functions of HRS if bad channels have been
- DoHrsCorrSP - Step10: Task which corrects HRS spectra from IF non-linearity errors.
- DoHrsCutBandEdges - Task which cuts the edges of the HRS sub-bands, according to the bandpass of the filter.
- DoHrsFFT - Step9: Task which applies a FFT processing on the correlation functions of HRS.
- DoHrsFreq - Step10: Task which computes the frequency of the sub-bands.
- DoHrsNorm - Step4: Task which normalizes the raw correlation functions of HRS.
- DoHrsOffsetPow - Step3: Task which computes the offset and power of HRS.
- DoHrsPowCorr - Step6: Task which corrects the non-linearity error of the power of HRS.
- DoHrsQDCFast - Step5a: Task which corrects roughly the quantization distortion of the correlation functions.
- DoHrsQDCFull - Step5b: Task which corrects the quantization distortion of the correlation functions.
- DoHrsSmooth - Step12: Task which applies a Hanning smoothing on the spectra, which is equivalent to a Hanning
- DoHrsSubbands - Step2: Task which extracts the HRS sub-bands from the HRS readout.
- DoHrsSymm - Step8: Task which symmetries the correlation function of HRS to prepare the FFT.
- DoHrsWindow - Step7: Task which applies a Hanning or Hamming windowing to the correlation functions of HRS,
- DoWbsBadPixels - Step3: Task that apply the masking pixel list.Configurable to select/unselect masks.
- DoWbsFreq - Step9: create frequency table for each scan.
- DoWbsNonlin - Step5: perform nonlinearity correction for even and odd WBS pixels.

DoWbsScanCount - Step1: Task for the correction of scan Count and for the

DoWbsSubbands - Step11: perform the splitInCcd() in all HifiSpectrumDataset contained in the HifiTimelineProduct.

DoWbsZero - Step7: Subtract appropriate zero spectra for each time step. If the zeros are

MkHrsBadChans - Step2b: Task which checks Internal Tests of HRS to find the bad channels.

MkWbsBadPixels - Step2: Task that check for saturated pixels

MkWbsFluxAtten - Step10: Attenuator settings analysis.

MkWbsFreq - Step8: Derive frequency scale from comb spectra.

MkWbsZero - Step6: Check the zeros and compute an interpolation function to

HIFI_user_task

ChangeDb - Allows change of database being used

HRS task

HrsCheckFtTask - Task which checks the Functional Tests of HRS.

HrsHKViewTask - Task which displays the values of all digital and analog HK values of HRS into a JTable. With

HrsQlaTask - Task which starts the HRS Quick Look Analysis (QLA).

Products

DatasetWrapper - DatasetWrapper is a (very) simple wrapper to transform a Dataset into

HifiProduct - HifiProduct is an extension of MapContext<-Product, which contains

generic task

ConverFrequencyTask - ConvertFrequencyTask

task

DataframeCount - A table dataset of the expected and actual dataframes after an observation has been

DisplayPacketTask - DisplayPacketTask is a task used to derive parameter values from an array of

HiClassTask - Export HIFI spectra to a FITS file that CLASS can read.

hrsPipelineTask - Process

IltObsContext - Retrieves the context in which an ILT observation was carried out.

IVviewer - Viewer of IVcurve Scans

level1PipelineTask - Process

level2PipelineTask - Process

MkSpur - A task to identify spurs in WBS HTPs

QualAssessHifiGenericTask - Task for quality assessment of level 1 Products

QualAssessHifiHrsTask - Task for quality assessment of level 1 Products.

QualAssessHifiProductTask - Task for quality assessment of level 0 Products

QualAssessHifiTask - Parent Task class for other quality assessment Tasks

QualAssessHifiWbsTask - Task for quality assessment of level 1 Products

WbsCheckFt - Housekeeping plotter

wbsPipelineTask - Process

How to use this manual

The *User's Reference Manual* contains information about all the main tasks and classes that you can use within your scripts. There are four version of this manual: the *HCSS* version describes the functions provided by the core Herschel software, and is always shipped with HIPE; the three other versions describe functions provided by HIFI, PACS and SPIRE software. You may or may not have all of these additional manuals, depending on the software you installed.

This manual includes the following sections:

- Categorized view of Commands. All the functions described in the manual, organised by category.
- Section 1.1. This section lists all the available commands in alphabetical order. Each command has a description, usage instructions and examples.

II.1. Related documentation

The aim of this manual is to provide reference information for all the user-relevant aspects of the HIFI software. Despite our efforts, you may find that some routines are missing, or have incomplete or inaccurate descriptions. In this case you are encouraged to consult the *Javadoc* developer's reference documentation. You can access the Javadoc by clicking on *HIFI Developer's Reference Manual (API)* in the table of contents of the HIPE Help System.

Guidance on how to use the Javadoc is provided in the *Scripting and Data Mining* guide: ????.

Some Javadoc pages may have links to more in-depth developer documentation. Be aware that these are *not* fully fledged help documents and are most useful to system developers or advanced users only.

Inspecting the source code

If you are an advanced user versed in Java and Jython, you might want to have a look at the source code of a task or class. You can include source code in your HIPE installation in the following ways:

- If you are installing a developer build via the Continuous Installation System, the `--src=yes` option will install source code. This is enabled by default if you use the `--developer` option.

  With the `--unpack=yes` option, the source code will be unpacked into a `src` subdirectory in your HIPE installation. With the `--unpack=no` option, source files for each module will be kept as ZIP files in the repository (usually under `$HOME/.hcss.d/repository` in UNIX systems and under `%HOMEPATH%/.hcss.d/repository` in Windows).

- If you are using an installer, select the checkbox next to the question *Would you like to have the source code installed?* This is only available if you choose the *Advanced* installation. The source code will be in the `src` subdirectory of your HIPE installation.

Chapter 1. DP Commands

1.1. Introduction

This chapter is a complete listing of all built-in DP functions, task and objects, collectively referred to as commands.

How to use this chapter

All of Dp's commands are documented alphabetically in this chapter. A first section gives a general overview what a command is used for and how to use it. This includes a general description and some examples. The "API summary" section provides a quick overview of the available constructors, methods and properties of a command. It is intended as quick reference for users that just need to remember a certain functionality. The "API details" section provides more detailed information about the constructors, methods and properties. Furthermore it provides links to other references and information about the command's history.

Overview

A table gives the full name of the command, indicates if it is written in Java or Jython and if it is just an ordinary command or if it follows the HCSS Task specifications. The import statement in this table can be copied to a Jython console.

Description

General description of the command.

Examples

Most commands include one or more examples that demonstrates how the command is used. Most of the examples are just one or two lines of DP code that can be entered at the Jython prompt. Others are code fragments or routines designed to serve as an examples for your own programs.

Limitations and Miscellaneous

The "Limitations" and "Miscellaneous" sections describes the boundaries of applicability and other specialties of the command.

API Summary

The API Summary provides a quick summary of the commands constructor, method, and properties. Note that most of the arguments are positional parameters that must be supplied in the order indicated by the command's syntax. However arguments can be often supplied by specifying their names. If its the case then they can provided in any order.

Constructor

Constructors are used to create a command.

Method

Methods are functions of a command that can be executed.

Properties

Properties are fields of a command that can be read or written by a command. INPUT properties contain values that are required by a command, OUTPUT properties contain the result of a command, INOUT provide both functionalities at the same time.

See also

The "See also" section provides links to related commands, to other manuals, or to external resources in the Internet.

History

The "History" section describes the changes occurred to the command.

1.2. AbstractScalarCorrection

Full Name:	herschel.hifi.pipeline.generic.AbstractScalarCorrection
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import AbstractScalarCorrection

Description

Base class for antenna temperature correction or beam efficiency correction.

1.3. AccessComponents

Full Name:	herschel.hifi.dp.access.gui.AccessComponents
Type:	Java Class -
Import:	from herschel.hifi.dp.access.gui import AccessComponents

Description

AccessComponents

Helper class for defining the GUI components for AccessDataFrameTask and AccessPacketTask.

History

- `2005-02-05 - pz`: First created.
- `2009-04-27 - ke`: adding database selector
- `2009-05-19 - ke`: Fixed cast error HIFI SPR-0647
- `2009-06-07 - pz`: updated to new Task.views parameter, i.e. removing HifiTask specifics
- `2009-06-28 - ke`: Enabled switching db's with out using the properties pop-up window
- `2009-07-24 - ke`: Enabled MibSupport for AccessPacketTask
- `2009-08-07 - ke`: Broke this class into three classes, originally called selector.java

1.4. AccessDataFrameComponents

Full Name:	herschel.hifi.dp.access.gui.AccessDataFrameComponents
Type:	Java Class -
Import:	from herschel.hifi.dp.access.gui import AccessDataFrameComponents

Description

AccessDataFrameCompenents

Helper class for defining the GUI components for AccessDataFrameTask.

History

- `2005-01-19 - pz`: First created.
- `2006-04-27 - ke`: Adding database selector
- `2006-05-19 - ke`: Fixed cast error HIFI SPR-0647
- `2006-06-07 - pz`: Updated to new Task.views parameter, i.e. removing HifiTask specifics
- `2006-06-28 - ke`: Enabled switching db's with out using the properties pop-up window
- `2006-07-24 - ke`: Enabled MibSupport for AccessPacketTask
- `2006-08-07 - ke`: Split Selector.java into three classes and renamed them.

1.5. AccessDataFrameTask

Full Name:	herschel.hifi.dp.access.AccessDataFrameTask
Alias:	AccessDataFrameTask
Type:	Java Task -
Import:	from herschel.hifi.dp.access import AccessDataFrameTask
Category:	DP Access

Description

DP Access Module: AccessDataFrameTask

Allows for access to HIFI DataFrames in a Versant database. It should be noted that one must remain connected to the database as the objects are not initially loaded in ones session until you actuall try and use them. Only references are returned initially. The loss of the connection to a database will result in not being able to use the objects from the database and may require you to restart your IA session.

Example

Example 1: AccessDataFrameTask

```
<pre>
from herschel.hifi.dp.access import AccessDataFrameTask
# usage on command line:
df = AccessDataFrameTask()(apids=[1030,1031],obsid=1342177771)
# usage on command line setting the database:
df = AccessDataFrameTask()(obsid1342177771=,
db="hifi_icc_ops_1@iccdb1.sron.rug.nl 0 READ")
# access dataframes using gui:
dftask = AccessDataFrameTask()
dftask.obsid=1342177771
dftask.gui=1
# here use gui to setup parameters and select -"execute"
df = dftask.result
# POSSIBLY use gui again for select selection, select -"execute"
df2 = dftask.result
# print the number of dataframes
print len(df)
# display dataframes, option 1:
dd = DisplayDataFrameTask()
dd.dataframe=df[0]
dd.gui = 1
# display dataframes, option 2:
browse_df = BrowseDfQuery()(gui=1)
# feedback dataframes back into your session using:
dfs = browse_df.query
df = browse_df.selected
# other options:
# to pass an existing store this task should reuse:
dfs = AccessDataFrameTask()(store=myStore, -.....)
</pre>
```

API Summary

Jython Syntax

```
see example below
```

Properties

```
Long bbtype [INPUT, OPTIONAL, default=null]
```

Properties
Long **obsid** [INPUT, OPTIONAL, default=null]
Integer **apid** [INPUT, OPTIONAL, default=null]
Integer[] **apids** [INPUT, OPTIONAL, default=null]
FineTime **start** [INPUT, OPTIONAL, default=null]
FineTime **end** [INPUT, OPTIONAL, default=null]
boolean **stream** [INPUT, OPTIONAL, default=false]
Boolean **exit** [INPUT, OPTIONAL, default=false]
String **db** [INPUT, OPTIONAL, default=null]
Boolean **gui** [INPUT, OPTIONAL, default=false]
Boolean **dialog** [INPUT, OPTIONAL, default=false]
ObjectStore **store** [INPUT, OPTIONAL, default=null]
DataFrame[] **result** [OUTPUT, OPTIONAL, default=null]
ProductReader **reader** [OUTPUT, OPTIONAL, default=null]

Limitations

The amount of memory on ones system governs how much data can be retrieved from a database

API details

Properties

Long bbtype [INPUT, OPTIONAL, default=null]

Provides the Building block id.

Long obsid [INPUT, OPTIONAL, default=null]

Provides the Observation id.

Integer apid [INPUT, OPTIONAL, default=null]

Provides the apid.

Integer[] apids [INPUT, OPTIONAL, default=null]

Provides an array of apids.

FineTime start [INPUT, OPTIONAL, default=null]

Provides the start time.

FineTime end [INPUT, OPTIONAL, default=null]

Provides the end time.

boolean stream [INPUT, OPTIONAL, default=false]

Provides streaming output (ProductReader).

Boolean exit [INPUT, OPTIONAL, default=false]

Provides a handle to exit the gui.

String db [INPUT, OPTIONAL, default=null]
Provides the name of the database to access.

Boolean gui [INPUT, OPTIONAL, default=false]
Allows to handle this task using a gui.

Boolean dialog [INPUT, OPTIONAL, default=false]
Allows to setup the parameters for this task using a GUI (only possible under jide_new).

ObjectStore store [INPUT, OPTIONAL, default=null]
Allows to pass an already existing ObjectStore.

DataFrame[] result [OUTPUT, OPTIONAL, default=null]
gives an array of dataframes collected from the database.

ProductReader reader [OUTPUT, OPTIONAL, default=null]
alternate output.

History

- 2005-02-21 - jcg: change doc format for help support.
- 2006-02-02 - pz: implemented HifiTask adding a dialog and gui
- 2006-04-11 - ke: throw runtime exception if gui is used with jide
- 2006-04-27 - ke: added tabbed DATA, TIME and DATABASE selector panel
- 2006-06-07 - PAZ: updated to new Task.views parameter by Peer, i.e. removing HifiTask specifics
- 2006-07-10 - ke: allowed some task parameters to contain null values
- 2006-10-20 - ke: added times database query string to speed up access from remote sites
- 2007-01-10 - PAZ: do a close in case after one has retrieved data from a database
- 2007-01-14 - dk: product reader as output
- 2008-03-13 - ke: Change gui behaviour and the store parameter

1.6. AccessHkParamComponents

Full Name:	herschel.hifi.dp.access.gui.AccessHkParamComponents
Type:	Java Class -
Import:	from herschel.hifi.dp.access.gui import AccessHkParamComponents

Description

AccessHkParamTask

Helper class for defining the GUI components for AccessHkParamTask.

History

- `2005-02-02 - pz`: First created.
- `2006-04-27 - ke`: Adding database selector.
- `2006-05-19 - ke`: Fixed cast error HIFI SPR-0647.
- `2006-06-07 - pz`: Updated to new Task.views parameter, i.e. removing HifiTask specifics.
- `2006-06-28 - ke`: Enabled switching db's with out using the properties pop-up window.
- `2006-07-24 - ke`: Enabled MibSupport for AccessPacketTask.
- `2006-08-07 - ke`: Split selector.java up into three classes, now called AccessPacketComponents.java.
- `2006-03-16 - ke`: Fixed plot null pointer errors, updated mibSupprt method.
- `2007-03-30 - ke`: Updated, new task parameters added
- `2008-01-13 - ke`: Change GUI behaviour

1.7. AccessHkParamTask

Full Name:	herschel.hifi.dp.access.AccessHkParamTask
Alias:	AccessHkParamTask
Type:	Java Task -
Import:	from herschel.hifi.dp.access import AccessHkParamTask
Category:	DP Access

Description

DP Access Module: AccessHkParamTask

Allow direct parameter retrieval from TmSourcePackets in the database(s).It should be noted that one must remain connected to the database as the objects are not initially loaded in ones session until you actuall try and use them. Only references are returned initially. The loss of the connection to a database will result in not being able to use the objects from the database and may require you to restart your IA session.

Example

Example 1: AccessHkParamTask

```
<pre>
# Usage: Command line
#
from herschel.hifi.dp.access import AccessHkParamTask
ahpt = AccessHkParamTask()
#
ahpt(db=ilt_fm_5_prop@iccdb.sron.rug.nl, obsid=268505167, apid=2017, params =
["FPU_room_temp", -"FPU_shutter_temp"])
composite=ahpt.paramresult
print composite
#
# or, only using one parameter
ahpt(obsid=268505167, apid=2017, param="FPU_room_temp")
# or, including a screenshot per 10 secs ( no averaging done -):
ahpt(obsid=268505167, apid=2017, param="FPU_room_temp", resolution=10)
# or, including more than one parameter
ahpt(obsid=268505167, apid=2017, params =
["FPU_room_temp", -"FPU_shutter_temp"], resolution=10)
# or, including a resolution for each parameter:
ahpt(obsid=268505167, apid=2017, params =
["FPU_room_temp", -"FPU_shutter_temp"], resolutions=[1,10])
# Translating your parameters into a query (i.e. apid -/ type -/ subtype -/
SID) such that the query returns only packets of interest
# This is introduced to constrain the number of packets on the client side to
lower the risk of memory problems:
ahpt(obsid=268505167, p2q=1, params = ["FPU_room_temp", -"FPU_shutter_temp"],
resolutions=[1,10])
#
# Usage: GUI
# In the gui select: database=ilt_fm_5_prop@iccdb.sron.rug.nl,
obsid=268505167, apid=2017, param=FPU_room_temp
#
ahpt = AccessHkParamTask()
ahpt.gui = 1
composite = ahpt.paramresult
print composite
print composite['FPU_room_temp'].getColumn("time_conv")
tbl1 = composite.get("FPU_room_temp")
PlotXY(tbl1.getColumn("time_conv").getData(),tbl1.getColumn("conv").getData())
</pre>
```

API Summary

Jython Syntax
see example below

Properties
Long **bbtype** [INPUT, OPTIONAL, default=null]
Long **obsid** [INPUT, OPTIONAL, default=null]
Integer **apid** [INPUT, OPTIONAL, default=null]
Integer[] **apids** [INPUT, OPTIONAL, default=null]
Integer **type** [INPUT, OPTIONAL, default=null]
Integer[] **types** [INPUT, OPTIONAL, default=null]
Integer[] **subtypes** [INPUT, OPTIONAL, default=null]
Integer **sid** [INPUT, OPTIONAL, default=null]
Integer[] **sids** [INPUT, OPTIONAL, default=null]
String **db** [INPUT, OPTIONAL, default=null]
FineTime **start** [INPUT, OPTIONAL, default=null]
FineTime **end** [INPUT, OPTIONAL, default=null]
Boolean **exit** [INPUT, OPTIONAL, default=false]
Boolean **gui** [INPUT, OPTIONAL, default=false]
Boolean **dialog** [INPUT, OPTIONAL, default=false]
String **param** [INPUT, OPTIONAL, default=null]
String[] **params** [INPUT, OPTIONAL, default=null]
Boolean **p2q** [INPUT, OPTIONAL, default=false]
Double **resolution** [INPUT, OPTIONAL, default=null]
double[] **resolutions** [INPUT, OPTIONAL, default=null]
ProductReader **reader** [OUTPUT, OPTIONAL, default=null]
CompositeDataset **paramresult** [OUTPUT, OPTIONAL, default=null]

Limitations

The amount of data that can be loaded is limited to size of ones java heap space

API details

Properties

Long bbtype [INPUT, OPTIONAL, default=null]
Provides the Building block id.

Long obsid [INPUT, OPTIONAL, default=null]
Provides the Observation id.

Integer apid [INPUT, OPTIONAL, default=null]
Provides the apid.

Integer[] apids [INPUT, OPTIONAL, default=null]

Provides an array of apids.

Integer type [INPUT, OPTIONAL, default=null]

Provides the type.

Integer[] types [INPUT, OPTIONAL, default=null]

Provides an array of types.

Integer[] subtypes [INPUT, OPTIONAL, default=null]

Provides an array of subtypes.

Integer sid [INPUT, OPTIONAL, default=null]

Provides the sid.

Integer[] sids [INPUT, OPTIONAL, default=null]

Provides an array of sids.

String db [INPUT, OPTIONAL, default=null]

Provides the name of the database to access.

FineTime start [INPUT, OPTIONAL, default=null]

Provides the start time.

FineTime end [INPUT, OPTIONAL, default=null]

Provides the end time.

Boolean exit [INPUT, OPTIONAL, default=false]

Provides a handle to exit gui.

Boolean gui [INPUT, OPTIONAL, default=false]

Allows to handle task using a gui.

Boolean dialog [INPUT, OPTIONAL, default=false]

Allows to define parameters using a GUI.

String param [INPUT, OPTIONAL, default=null]

Provides a single parameter id.

String[] params [INPUT, OPTIONAL, default=null]

Provides an array of parameter ids.

Boolean p2q [INPUT, OPTIONAL, default=false]

p2q = Boolean.TRUE will translate the parameters into a specific query only returning those packets which contain the requested parameter. This is used when searching large numbers of packets were the amount of computer memory is an issue. It has been shown that setting this can slow down database access for remote users.

Double resolution [INPUT, OPTIONAL, default=null]

Provides a way to set the resolution in secs for a single parameter (or more -, as being passed by the taskparameter: params).

double[] resolutions [INPUT, OPTIONAL, default=null]

Provides a way to set the resolution in secs for each param as passed by the taskparameter: params .

ProductReader reader [OUTPUT, OPTIONAL, default=null]

Alternate output for database query access. A reader for a stream of HK Packets

CompositeDataset paramresult [OUTPUT, OPTIONAL, default=null]

Returns an composite containing a tabledataset containing parameter values passed for the packets collected from the database.

1.8. AccessPacketComponents

Full Name:	herschel.hifi.dp.access.gui.AccessPacketComponents
Type:	Java Class -
Import:	from herschel.hifi.dp.access.gui import AccessPacketComponents

Description

AccessPacketTask

Helper class for defining the GUI components for AccessPacketTask.

History

- `2005-02-02 - pz`: First created.
- `2006-04-27 - ke`: Adding database selector
- `2006-05-19 - ke`: Fixed cast error HIFI SPR-0647
- `2006-06-07 - pz`: Updated to new Task.views parameter, i.e. removing HifiTask specifics
- `2006-06-28 - ke`: Enabled switching db's with out using the properties pop-up window
- `2006-07-24 - ke`: Enabled MibSupport for AccessPacketTask
- `2006-08-07 - ke`: Split selector.java up into three classes, now called AccessPacketComponents.java.

1.9. AccessPacketTask

Full Name:	herschel.hifi.dp.access.AccessPacketTask
Alias:	AccessPacketTask
Type:	Java Task -
Import:	from herschel.hifi.dp.access import AccessPacketTask
Category:	DP Access

Description

DP Access Module: AccessPacketTask

Allows access to TmSourcePackets in the database.It should be noted that one must remain connected to the database as the objects are not initially loaded in ones session until you actuall try and use them. Only references are returned initially. The loss of the connection to a database will result in not being able to use the objects from the database and may require you to restart your IA session.

Example

Example 1: AccessPacketTask

```
<pre>
from herschel.hifi.dp.access import AccessPacketTask
# usage on command line:
pk = AccessPacketTask()(apids=[1030,1026],obsid=1342177771)
# usage on command line setting the database:
pk = AccessPacketTask()(obsid=1342177771,
db="hifi_icc_ops_1@iccdb1.sron.rug.nl 0 READ")
# access packets using gui:
hktask = AccessPacketTask()
hktask.obsid=1342177771
hktask.gui=1
# use gui to setup parameters and select -"execute"
pk = hktask.result
# POSSIBLY use gui again for select selection, select -"execute"
pk2 = hktask.result
# print the number of packets
print len(pk)
# other options:
# to pass an existing store this task should reuse:
pk = AccessPacketTask()(store=myStore, -.....)
# Other related tasks: for extracting specific parameters use:
hk_table = AccessHkParamTask()(params=[p1,p2, -.....], resolutions=[1,10,....])
# Note: also for these tasks GUI are provided to allow browsing
</pre>
```

API Summary

Properties
Long **bbtype** [INPUT, OPTIONAL, default=null]
Long **obsid** [INPUT, OPTIONAL, default=null]
Integer **apid** [INPUT, OPTIONAL, default=null]
Integer[] **apids** [INPUT, OPTIONAL, default=null]
Integer **type** [INPUT, OPTIONAL, default=null]
Integer[] **types** [INPUT, OPTIONAL, default=null]

Properties
Integer **subtype** [INPUT, OPTIONAL, default=null]
Integer[] **subtypes** [INPUT, OPTIONAL, default=null]
Integer **sid** [INPUT, OPTIONAL, default=null]
Integer[] **sids** [INPUT, OPTIONAL, default=null]
String **param** [INPUT, OPTIONAL, default=null]
String[] **params** [INPUT, OPTIONAL, default=null]
String **packettype** [INPUT, OPTIONAL, default=null]
String[] **packettypes** [INPUT, OPTIONAL, default=null]
Boolean **p2q** [INPUT, OPTIONAL, default=false]
String **db** [INPUT, OPTIONAL, default=No default value]
FineTime **start** [INPUT, OPTIONAL, default=null]
FineTime **end** [INPUT, OPTIONAL, default=null]
boolean **stream** [INPUT, OPTIONAL, default=false]
Boolean **exit** [INPUT, OPTIONAL, default=false]
Boolean **gui** [INPUT, OPTIONAL, default=false]
Boolean **dialog** [INPUT, OPTIONAL, default=false]
ObjectStore **store** [INPUT, OPTIONAL, default=false]
TmSourcePacket[] **result** [OUTPUT, OPTIONAL, default=null]
ProductReader **reader** [OUTPUT, OPTIONAL, default=null]

Limitations

The amount of data that can be loaded is limited to size of ones java heap space

API details

Properties

Long bbtype [INPUT, OPTIONAL, default=null]

Provides the Building block id.

Long obsid [INPUT, OPTIONAL, default=null]

Provides the Observation id.

Integer apid [INPUT, OPTIONAL, default=null]

Provides the apid.

Integer[] apids [INPUT, OPTIONAL, default=null]

Provides an array of apids.

Integer type [INPUT, OPTIONAL, default=null]

Provides the type.

Integer[] types [INPUT, OPTIONAL, default=null]

Provides an array of types.

Integer subtype [INPUT, OPTIONAL, default=null]

Provides the subtype.

Integer[] subtypes [INPUT, OPTIONAL, default=null]

Provides an array of subtypes.

Integer sid [INPUT, OPTIONAL, default=null]

Provides the sid.

Integer[] sids [INPUT, OPTIONAL, default=null]

Provides an array of sids.

String param [INPUT, OPTIONAL, default=null]

Provides a single parameter id.

String[] params [INPUT, OPTIONAL, default=null]

Provides an array of parameter ids.

String packettype [INPUT, OPTIONAL, default=null]

Provides a single packettype.

String[] packettypes [INPUT, OPTIONAL, default=null]

Provides an array of packettypes.

Boolean p2q [INPUT, OPTIONAL, default=false]

p2q = Boolean.TRUE will translate the parameters into a specific query only returning those packets which contain the requested parameter. This is used when searching large numbers of packets were the amount of computer memory is an issue. It has been shown that setting this can slow down database access for remote users.

String db [INPUT, OPTIONAL, default=No default value]

Provides the name of the database to access.

FineTime start [INPUT, OPTIONAL, default=null]

Provides the start time.

FineTime end [INPUT, OPTIONAL, default=null]

Provides the end time.

boolean stream [INPUT, OPTIONAL, default=false]

Provides streaming output (ProductReader).

Boolean exit [INPUT, OPTIONAL, default=false]

Provides a handle to exit gui.

Boolean gui [INPUT, OPTIONAL, default=false]

Allows to handle task using a gui.

Boolean dialog [INPUT, OPTIONAL, default=false]
Allows to define parameters using a GUI.

ObjectStore store [INPUT, OPTIONAL, default=false]
Allows to pass an already existing ObjectStore.

TmSourcePacket[] result [OUTPUT, OPTIONAL, default=null]
Returns an array of packets collected from the database.

ProductReader reader [OUTPUT, OPTIONAL, default=null]
Alternate output for database query access. A reader for a stream of HK Packets

History

- 2005-02-21 - jcg: change doc format for help support.
- 2006-02-02 - pz: implemented HifiTask adding a dialog and gui
- 2006-04-11 - ke: throw runtime exception if gui is used with jide
- 2006-04-27 - ke: added tabbed DATA, TIME and DATABASE selector panel
- 2006-06-07 - PAZ: updated to new Task.views parameter by Peer, i.e. removing HifiTask specifics
- 2006-07-10 - ke: allowed some task parameters to contain null values
- 2006-07-24 - ke: integrated MibSupport into task
- 2006-10-23 - ke: added times to all queries with an obsid (speed up remote access to dbs)
- 2007-01-10 - PAZ: do a close in case after one has retrieved data from a database
- 2007-01-25 - DK: add a streaming option and output
- 2007-02-16 - ke: reusing the store from Access and moved the code that adds times to the db queries
- 2007-03-29 - ke: adding new parameters for export for the non-gui part of this application.
- 2007-03-30 - ke: refactoring the execute method and created a new method call setUpLocal-Connection
- 2008-03-13 - ke: Changes to the gui behaviour and to the store parameter

1.10. AverageSpectrum

Full Name:	herschel.hifi.dp.dataset.spectrum.AverageSpectrum
Type:	Java Task -
Import:	from herschel.hifi.dp.dataset.spectrum import AverageSpectrum

Description

Task for averaging the spectra included in a timeline product or a Spectrum2d.

By default, a {@link Spectrum2d} dataset is passed even if a timeline product has been specified as input. Then, the original input product remains unchanged. If the flag 'return_single_ds' is set to 'False' the task is processed on a per dataset basis if a timeline product is specified. In this case, the (selected) datasets are processed on an individual basis and the original datasets included in the timeline product are replaced by the datasets resulting from the averaging procedure.

Different selection schemes are available for configuring what averages should be calculated (selection of datasets from timeline products and/or rows from datasets; specification of 'groups' of rows for which the average should be calculated on an individual per group basis. See the task {@link herschel.ia.toolbox.spectrum.AverageSpectrum} the functionality available for datasets is inherited from and {@link herschel.hifi.dp.dataset.spectrum.SpectrumTask}.

API Summary

Jython Syntax

```
from herschel.hifi.dp.dataset.spectrum import AverageSpectrum
avgHifi = AverageSpectrum()
avgSpectra = avgHifi(htp=product, selection_meta={"sds_type":
["science"]})
avgSpectra = avgHifi(htp=product, selection_meta={"sds_type":
["science"]}, return_single_ds=False)
avgSpectra = avgHifi(htp=product, selection_table={"bbtype":
[6031]})
avgSpectra = avgHifi(htp=product, selection_lookup={"bbtype":
[6031]})
avgSpectra = avgHifi(ds=spectrum, selection_index=[1,7,9,20]})
```

Properties

HifiTimelineProduct **htp** [INOUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds1** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds2** [INPUT, OPTIONAL, default=no default value.]
PyDictionary **selection_meta** [INPUT, OPTIONAL, default=no default value.]
Boolean **return_single_ds** [INPUT, OPTIONAL, default=False.]
SelectionModel **selection** [INPUT, OPTIONAL, default=no default value.]
PyDictionary|Map> **selection_lookup** [INPUT, OPTIONAL, default=no default value.]

Properties
PyList **selection_index** [INPUT, OPTIONAL, default=No default value.]
String **variant** [INPUT, OPTIONAL, default=no default value.]
Double **param** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]
Boolean **per_group** [INPUT, OPTIONAL, default=no default value.]
GroupingModel **grouping** [INPUT, OPTIONAL, default=no default value.]

API details

Properties

HifiTimelineProduct htp [INOUT, OPTIONAL, default=no default value.]

The timeline product to be processed.

SpectrumContainer ds [INPUT, OPTIONAL, default=no default value.]

Input container to be processed by the task: in case the task is used as a many-to-one operation (eg avg) or as scalar operation.

SpectrumContainer ds1 [INPUT, OPTIONAL, default=no default value.]

First input container for pair-wise operations.

SpectrumContainer ds2 [INPUT, OPTIONAL, default=no default value.]

Second input container for pair-wise operations.

PyDictionary selection_meta [INPUT, OPTIONAL, default=no default value.]

PyDictionary with the selection criteria based on meta data.

Boolean return_single_ds [INPUT, OPTIONAL, default=False.]

Flag indicating whether a single dataset or a hifi timeline product should be returned.

SelectionModel selection [INPUT, OPTIONAL, default=no default value.]

Specify an instance of any kind of SelectionModel here.

PyDictionary|Map> selection_lookup [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated.

PyList selection_index [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered.

String variant [INPUT, OPTIONAL, default=no default value.]

Specify the variant of processing mode you would like to run (including flags / weights / ...). HCSS defaults include: "flux" / "flux-wave" / "flux-wave-weight" / "flux-flag-wave" / "flux-weight-flag-wave"

Double param [INPUT, OPTIONAL, default=no default value.]

The scalar parameter to be considered when the task is used as scalar operation.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

Boolean per_group [INPUT, OPTIONAL, default=no default value.]

Specify that the average should be calculated on a per group basis - once a selection model is specified that provides a natural grouping of the data.

GroupingModel grouping [INPUT, OPTIONAL, default=no default value.]

Grouping model that can be used to partition the point spectra into groups and calculate the average on a per group basis. When a grouping model is specified, the 'per_group'-flag is obsolete.

History

- 01-June-2007 initial.
- 13-July-2007 Javadoc updated.
- 17-Aug-2007 Extended and refactored.

1.11. AverageSpectrum

Full Name:	herschel.hifi.pipeline.util.tools.AverageSpectrum
Type:	Java Task -
Import:	from herschel.hifi.pipeline.util.tools import AverageSpectrum

Description

Task for averaging the spectra included in a timeline product or a Spectrum2d.

By default, a {@link Spectrum2d} dataset is passed even if a timeline product has been specified as input. Then, the original input product remains unchanged. If the flag 'return_single_ds' is set to 'False' the task is processed on a per dataset basis if a timeline product is specified. In this case, the (selected) datasets are processed on an individual basis and the original datasets included in the timeline product are replaced by the datasets resulting from the averaging procedure.

Different selection schemes are available for configuring what averages should be calculated (selection of datasets from timeline products and/or rows from datasets; specification of 'groups' of rows for which the average should be calculated on an individual per group basis. See the task {@link herschel.ia.toolbox.spectrum.AverageSpectrum} the functionality available for datasets is inherited from and {@link herschel.hifi.dp.dataset.spectrum.SpectrumTask}.

API Summary

Jython Syntax

```
from herschel.hifi.dp.dataset.spectrum import AverageSpectrum
avgHifi = AverageSpectrum()
avgSpectra = avgHifi(htp=product, selection_meta={"sds_type":
["science"]})
avgSpectra = avgHifi(htp=product, selection_meta={"sds_type":
["science"]}, return_single_ds=False)
avgSpectra = avgHifi(htp=product, selection_lookup={"bbtype":
[6031]})
avgSpectra = avgHifi(htp=product, selection_meta={"sds_type":
["science"]}, selection_lookup={"bbtype":[6031]},
return_single_ds=True)
avgSpectra = avgHifi(ds=spectrum, selection_index=[1,7,9,20]})
```

Properties

HifiTimelineProduct **htp** [INOUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds1** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **ds2** [INPUT, OPTIONAL, default=no default value.]
PyDictionary **selection_meta** [INPUT, OPTIONAL, default=no default value.]
Boolean **return_single_ds** [INPUT, OPTIONAL, default=False.]
SelectionModel **selection** [INPUT, OPTIONAL, default=no default value.]

Properties
PyDictionary|Map> **selection_lookup** [INPUT, OPTIONAL, default=no default value.]
PyList **selection_index** [INPUT, OPTIONAL, default=No default value.]
String **variant** [INPUT, OPTIONAL, default=no default value.]
Double **param** [INPUT, OPTIONAL, default=no default value.]
SpectrumContainer **result** [OUTPUT, OPTIONAL, default=no default value]
Boolean **per_group** [INPUT, OPTIONAL, default=no default value.]
GroupingModel **grouping** [INPUT, OPTIONAL, default=no default value.]

API details

Properties

HifiTimelineProduct htp [INOUT, OPTIONAL, default=no default value.]

The timeline product to be processed.

SpectrumContainer ds [INPUT, OPTIONAL, default=no default value.]

Input container to be processed by the task: in case the task is used as a many-to-one operation (eg avg) or as scalar operation.

SpectrumContainer ds1 [INPUT, OPTIONAL, default=no default value.]

First input container for pair-wise operations.

SpectrumContainer ds2 [INPUT, OPTIONAL, default=no default value.]

Second input container for pair-wise operations.

PyDictionary selection_meta [INPUT, OPTIONAL, default=no default value.]

PyDictionary with the selection criteria based on meta data.

Boolean return_single_ds [INPUT, OPTIONAL, default=False.]

Flag indicating whether a single dataset or a hifi timeline product should be returned.

SelectionModel selection [INPUT, OPTIONAL, default=no default value.]

Specify an instance of any kind of SelectionModel here.

PyDictionary|Map> selection_lookup [INPUT, OPTIONAL, default=no default value.]

Specify a PyDictionary with keys given by the attribute name to look up and a list of admissible values. Behind the scenes, a DiscreteValueSelectionModel is instantiated.

PyList selection_index [INPUT, OPTIONAL, default=No default value.]

Specify a PyList with the indices of the point spectra to be considered.

String variant [INPUT, OPTIONAL, default=no default value.]

Specify the variant of processing mode you would like to run (including flags / weights / ...). HCSS defaults include: "flux" / "flux-wave" / "flux-wave-weight" / "flux-flag-wave" / "flux-weight-flag-wave"

Double param [INPUT, OPTIONAL, default=no default value.]

The scalar parameter to be considered when the task is used as scalar operation.

SpectrumContainer result [OUTPUT, OPTIONAL, default=no default value]

Result object containing the results of the operation applied.

Boolean per_group [INPUT, OPTIONAL, default=no default value.]

Specify that the average should be calculated on a per group basis - once a selection model is specified that provides a natural grouping of the data.

GroupingModel grouping [INPUT, OPTIONAL, default=no default value.]

Grouping model that can be used to partition the point spectra into groups and calculate the average on a per group basis. When a grouping model is specified, the 'per_group'-flag is obsolete.

History

- 01-June-2007 initial.
- 13-July-2007 Javadoc updated.
- 17-Aug-2007 Extended and refactored.

1.12. BrowseDf

Full Name:	herschel.hifi.dp.access.BrowseDf
Alias:	BrowseDf
Type:	Jython Task -
Import:	from herschel.hifi.dp.access import BrowseDf

Description

The BrowseDf task is used to display dataframes data associated

with an observation stored in any accessable database. To change to a new database type the name of the database into the local database text field located on the database tab. The fastest way to access data from the database is to lookup data using a specific obsid. If the times in the time selector panel are the same the script will automatically get the begin and end time of the observation and add it to the query string. If the times are different this will override the automatic process and search the database based on those times

The execute button is used to aquire data from the database. The clear button will clear all dataframes from memory and remove them from the selection box

Example

Example 1: BrowseDf

```
<pre>
# Interactive mode (Database: ilt_fm_2_prop obsid: 268435480)
from herschel.hifi.dp.access.BrowseDf import *
dfBrowser = BrowseDf()
dfBrowser.gui = 1
# This only brings up the browser. Once selections are made, and the
# execute button is pressed, then one can export df(s) in their
# session
#
# resulting_dfs = dfBrowser.query
# selected_df = dfBrowser.selected
#
# The task also also be pre-initialized from the command-line
# dfBrowser = BrowseDf()(db="ilt_fm_2_prop@iccdb1.sron.rug.nl 0 READ",
obsid=268435480)
# or
# dfBrowser = BrowseDf()
# dfBrowser.db = -"ilt_fm_2_prop@iccdb1.sron.rug.nl 0 READ"
# dfBrowser.obsid = 268435480
# dfBrowser()
</pre>
```

API Summary

Properties
Array(Dataframe) **query** [OUTPUT, OPTIONAL, default=None]
Boolean **gui** [INPUT, OPTIONAL, default=false]
Dataframe **selected** [OUTPUT, OPTIONAL, default=None]
Long **obsid** [INPUT, OPTIONAL, default=None]
Integer **apid** [INPUT, OPTIONAL, default=None]
Integer[] **apids** [INPUT, OPTIONAL, default=None]

Properties
Long **bbtype** [INPUT, OPTIONAL, default=None]
String **db** [INPUT, OPTIONAL, default=None]

API details

Properties

Array(Dataframe) query [OUTPUT, OPTIONAL, default=None]

This is the array of dataframe returned from the query to the database

Boolean gui [INPUT, OPTIONAL, default=false]

When gui = true a GUI will show up to allow input by user-friendly swing components. Note: This option is currently forced true

Dataframe selected [OUTPUT, OPTIONAL, default=None]

This is the user selected dataframe

Long obsid [INPUT, OPTIONAL, default=None]

This is the observation id

Integer apid [INPUT, OPTIONAL, default=None]

This is a single APID of an Observation

Integer[] apids [INPUT, OPTIONAL, default=None]

This is an array of APIDs of an Observation

Long bbtype [INPUT, OPTIONAL, default=None]

This is a BBType of an Observation

String db [INPUT, OPTIONAL, default=None]

Point to a different database other than default

History

- 2006-04-11 - KE: Created
- 2006-05-11 - KE: Updated with the new gui plot
- 2006-05-22 - KE: Fixed documentation format
- 07-06-06 PAZ: update to Task.views API
- 2006-06-22 - KE: Clear dataframe from memory
- 2006-06-28 - KE: Enable changing the database without using pop-up window
- 2007-06-29 - KE: Removing gui components from constructor
- 2007-11-12 - KE: Added additional task parameters from AccessDataFrameTask
- 2009-04-15 - KE: Updated to reflect changes in IA_TASK - TaskParameter Flushing

1.13. BrowseHk

Full Name:	herschel.hifi.dp.access.BrowseHk
Alias:	BrowseHk
Type:	Jython Task -
Import:	from herschel.hifi.dp.access import BrowseHk

Description

The BrowseHK task is used to display House Keeping data associated

with an observation stored in any accessable database. To change to a new database type the name of the database into the local database text field located on the database tab. The fastest way to access data from the database is to lookup data using a specific obsid. If the times in the time selector panel are the same the script will automatically get the begin and end time of the observation and add it to the query string. If the times are different this will override the automatic process and search the database based on those times

The execute button is used to aquire data from the database. The exit button is used to close the application.

Example

Example 1: BrowseHk

```
<pre>
# interactive mode (Database: ilt_fm_2_prop obsid: 268435480)
from herschel.hifi.dp.acccess.BrowseHk import *
hkBrowser = BrowseHk()
hkBrowser,gui = 1
# This only brings up the browser. Once selections are made, and the
# execute button is pressed, then one can export pk-params(s) in their
# session
#
# resulting_pks = hkBrowser.query  # array[TmSourcePacket]!!
# hkBrowser.selectedFlag = 1       # Trigger the output of selected
parameters from the gui
# selected_pk = hkBrowser.selected # TableDataset!!
</pre>
```

API Summary

Properties
Array(TmSourcePacket) **query** [OUTPUT, OPTIONAL, default=None]
Boolean **gui** [INPUT, OPTIONAL, default=false]
Boolean **selectedFlag** [INPUT, OPTIONAL, default=false]
TableDataset **selected** [OUTPUT, OPTIONAL, default=None]

API details

Properties

Array(TmSourcePacket) query [OUTPUT, OPTIONAL, default=None]

This is the array of TmSourcePackets (House Keeping data) returned from the query to the database

Boolean gui [INPUT, OPTIONAL, default=false]

When gui = true a GUI will show up to allow input by user-friendly swing components. Note: This option is currently forced true

Boolean selectedFlag [INPUT, OPTIONAL, default=false]

When selectedFlag = true This triggers the output of selected parameters from the gui to the output taskparameter 'selected'.

TableDataset selected [OUTPUT, OPTIONAL, default=None]

This is the generated TableDataset of the House Keeping data

History

- 2006-04-10 - KE: Created
- 2006-05-22 - KE: Fixed documentation format
- 2006-06-07 - PZ: update to Task.views API
- 2006-06-28 - KE: Enable changing the database without using pop-up window
- 2006-08-08 - KE: Updated for use with gui plot
- 2007-06-29 - KE: Removing gui components from constructor
- 2009-04-15 - KE: Updated to reflect changes in IA_TASK - TaskParameter Flushing

1.14. CalFluxHotCold

Full Name:	herschel.hifi.pipeline.generic.CalFluxHotCold
Type:	Java Class -
Import:	from herschel.hifi.pipeline.generic import CalFluxHotCold

Description

Class containing receiver temperature and bandpass information of an observation.

Several datasets are included in the Hifi product:

- Receiver temperature and band pass are in different datasets.
- In case different LO frequencies are measured the corresponding calibration data is written to different datasets.

The product manages the referencing of the datasets within the product: Receiver temperature have uneven and bandpass have even integer keys (starting with 1).

Within each dataset, possibly several rows are included. Each row is associated with a different observation time. The observation times are used for returning suitably interpolated values that allow to rescale the original source flux data to the physical intensity scale.

1.15. CalHrsBadChans

Full Name:	herschel.hifi.cal.hrs.CalHrsBadChans
Type:	Java Class - 
Import:	from herschel.hifi.cal.hrs import CalHrsBadChans

Description

Product which contains the bad channels table for HRS.

See Documentation "380-HRS_Calibration_Products-V1.pdf".

1.16. CalHrsPowCorr

Full Name:	herschel.hifi.cal.hrs.CalHrsPowCorr
Type:	Java Class -
Import:	from herschel.hifi.cal.hrs import CalHrsPowCorr

Description

Product which contains the values for correcting the power gain non-linearity of HRS.

See Documentation "380-HRS_Calibration_Products-V1.pdf".

1.17. CalHrsQDCFast

Full Name:	herschel.hifi.cal.hrs.CalHrsQDCFast
Type:	Java Class -
Import:	from herschel.hifi.cal.hrs import CalHrsQDCFast

Description

Product which contains the Fast Quantization Distortion Correction factor for HRS.

See Documentation "380-HRS_Calibration_Products-V1.pdf".

1.18. CalHrsQDCFull

Full Name:	herschel.hifi.cal.hrs.CalHrsQDCFull
Type:	Java Class -
Import:	from herschel.hifi.cal.hrs import CalHrsQDCFull

Description

Product which contains the Full Quantization Distortion Correction tables for HRS.

See Documentation "380-HRS_Calibration_Products-V1.pdf".

1.19. CalOffBaseline

Full Name:	herschel.hifi.pipeline.generic.CalOffBaseline
Type:	Java Class -
Import:	from herschel.hifi.pipeline.generic import CalOffBaseline

Description

Product containing the baseline spectra obtained by by the MkOffSmooth module which processes

off data sets.

The data is written to datasets of type {@link Spectrum2d} - for each frequency group a different dataset is created.

1.20. CalPhases

Full Name:	herschel.hifi.pipeline.generic.CalPhases
Type:	Java Class -
Import:	from herschel.hifi.pipeline.generic import CalPhases

Description

Product that information about the different phases observed with an observation.

Phases are identified (depending on the observing mode) from the Chopper / buffer or the LoFrequency / buffer.

The information is found in a suitable summary table.

1.21. CalWbsFreqCoeff

Full Name:	herschel.hifi.cal.wbs.CalWbsFreqCoeff
Type:	Java Class -
Import:	from herschel.hifi.cal.wbs import CalWbsFreqCoeff

Description

Use it as a table dataset.

See the javadoc API for details.

1.22. CalWbsFreq

Full Name:	herschel.hifi.cal.wbs.CalWbsFreq
Type:	Java Class - 
Import:	from herschel.hifi.cal.wbs import CalWbsFreq

Description

Use it as a table dataset.

See the javadoc API for details.

1.23. ChangeDb

Full Name:	herschel.hifi.pipeline.spg.plugin.ChangeDb
Type:	Jython Task -
Import:	from herschel.hifi.pipeline.spg.plugin import ChangeDb
Category:	HIFI_user_task

Description

Allows change of database being used

Tool to show and change the setting for the database in use

Example

Example 1: usage of ChangeDb

```
from herschel.hifi.scripts.users.share.ChangeDb import *
# Switch to new data base
ChangeDb()("sovt2_fm_4_prop",host="iccdb1.sron.rug.nl")
# Extract some WBS data
from herschel.hifi.scripts.users.volker.ReadObservation import *
htp = ReadObservation()(obsid=3221226273L,spectrometer="WBS-H")
```

API Summary

Jython Syntax

```
ChangeDb()(db = newdb, host = newhost)
```

Properties

`String db [INPUT, OPTIONAL, default=""]`

`String host [INPUT, OPTIONAL, default=Configuration.getProperty("var.database.server")]`

API details

Properties

`String db [INPUT, OPTIONAL, default=""]`

Database name. If no data base is specified, the current data base setting is displayed.

`String host [INPUT, OPTIONAL, default=Configuration.getProperty("var.database.server")]`

Hostname

History

- `2006-04-11 - TM`: created as a task
- `2006-06-30 - VO`: modified to include host and display of current setting
- `2007-02-14 - VO`: cleared up signatures to be compliant with latest API

- 2008-07-04 - KE: Removed import statements, code clean-up
- 2009-03-05 - VO: Switch to var.database.server property for default host
- 2009-04-23 - VO: Work-aoround for @ in var.database.server in user build

1.24. CheckDataStructure

Full Name:	herschel.hifi.pipeline.generic.CheckDataStructure
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import CheckDataStructure

Description

Check the datasets included in input the timeline product passed for

having unique bbnumber. If necessary the datasets are split. Otherwise, the input timeline product is returned.

Finally, the summary table is updated and additional information is written to it:

- "isLine": whether the science data contains the signal or just a OFF reference.
- "bbtype": the unique bbtype found in the datasets.
- "LoF": the single LoFrequency or the lower LoFrequency in F-Switch observations.
- "F-Throw": in F-Switch observations the F-Switch throw.

Example

Example 1: from herschel.hifi.pipeline.generic import CheckDataStructure

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

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default value]
HifiTimelineProduct **htpout** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

HifiTimelineProduct htpout [OUTPUT, MANDATORY, default=no default value]

The timeline product provided as output.

1.25. CheckFreqGrid

Full Name:	herschel.hifi.pipeline.generic.CheckFreqGrid
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import CheckFreqGrid

Description

Provide information about the frequency ranges observed in the timeline product and forms groups of datasets with common LO settings.

Furthermore, it computes suitable characteristics for measuring drifts in the grid of com measurements. This information is provided in form of a quality product (of type {@link FreqRanges}).

Two scans are comparable, if

- Number of subbands is identical.
- LO settings are comparable.

Comparable datasets are grouped. For each group, a {@link TableDataset} is included in the {@link FreqRanges}-product. These datasets contain the keys of dataset that belong to the same group as meta data, the grid of sub-sequent comb-frequencies within the given group and the drift-measure between these comb-measurements. Care is needed when referencing the dataset keys in the meta data in case the datasets in the original timeline product are rearranged.

The drift measure is calculated by

- first, computing for each subband the average difference between corresponding channel-frequencies of the sub-sequent comb measurements and
- second, taking the maximum over the absolute values of the per subband measures computed in the first step.
- Finally, this distance is divided by teh distiance in observation time.

The drift measure is reported in units Hz / sec.

Assumptions:

- Within each dataset, the frequency ranges are comparable. This step is resolved by the CheckDataStructure module.
- Datasets with comparable data are subsequent in time.
- In case of a FSwitch observation, it is assumed that in each dataset exactly two LoFrequencies are found.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import CheckFreqGrid
checkFreqGrid = CheckFreqGrid()
freqDrift = checkFreqGrid(htp=htp, tolerance=10)
```

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default value]
Double **tolerance** [INPUT, OPTIONAL, default=no default value]
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]
Boolean **isFSwitch** [INPUT, OPTIONAL, default=no default value]
FreqRanges **drift** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default value]

The timeline product (observation) to be analyzed.

Double tolerance [INPUT, OPTIONAL, default=no default value]

Tolerance (in Hz/sec) for the drift between successive comb measurements. If the tolerance is exceeded a warning is reported.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Configuration parameter that can be passed to the product

Boolean isFSwitch [INPUT, OPTIONAL, default=no default value]

Flag to indicate whether the given data should be treated as a FSwitch observing mode.

FreqRanges drift [OUTPUT, OPTIONAL, default=no default value]

The calibration/quality product provided as output.

History

- 2007-11-29 - meli: Initial version
- 2008-04-14 - meli: Bug fixes associated with the LO frequencies, signature extended by params
- 2008-05-16 - meli: New distance measure (--> actually a drift), PipelineConfiguration params

1.26. CheckPhases

Full Name:	herschel.hifi.pipeline.generic.CheckPhases
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import CheckPhases

Description

Check phases module analyses for the science and the hot/cold datasets whether the data

in the columns Chopper, LoFrequency and buffer follow specific patterns prescribed by the observing modes. For DBS and FastDBS modes the pointing information is also analyzed.

The module checks the columns for the patterns ABBA, ABAB, CONST. Further information is provided on what values are observed these columns and what values the sequences start with. The information is written to a table dataset: For each dataset, the information is written into one row.

Depending on the observing mode, the following patterns are expected (on a per dataset basis):

- Position Switch: Chopper, buffer, LoFrequency: CONST
- DBS: Chopper and buffer: ABBA, LoFrequency: CONST
- FastDBS: Chopper and buffer: ABAB, LoFrequency: CONST
- FSwitch: LoFrequency and buffer: ABBA, Chopper: CONST
- LoadChop: Chopper and buffer: ABBA, LOFrequency: CONST

If the observing mode cannot be identified, no logs are created.

Suitable log messages are created. Suitable quality flags are added to the meta data of the datasets (by using HifiQualityEnum's).

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import CheckPhases
checkPhases = CheckPhases()
calPhases = checkPhases(htp=htp)
```

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default Value]
CalPhases **cal** [OUTPUT, OPTIONAL, default=No default Value]
MapContext **calibration** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default Value]

The timeline product (observation) to be passed to the module.

CalPhases cal [OUTPUT, OPTIONAL, default=No default Value]
Calibration product including information about patterns found in the houskeeping data.

MapContext calibration [INPUT, OPTIONAL, default=no default value]
Parameter to pass a calibration context from which all the calibration input, here the chopper positions, can be retrieved.

History

- `2007-11-29 - meli`: initial version.
- `2007-04-02 - meli`: CalPhases added.
- `2007-04-28 - meli`: Info table extended, generation of table separated from generating the logs.
- `2009-01-08 - meli`: Quality information added to the meta data using HifiQualityEnum

1.27. ConverFrequencyTask

Full Name:	herschel.hifi.pipeline.product.ConvertFrequencyTask
Alias:	ConverFrequencyTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline.product import ConvertFrequencyTask
Category:	generic task

Description

ConvertFrequencyTask

Task to convert the frequencies in a HifiTimelineProduct (HTP) or HifiSpectrumDataset (SDS) to upper or lower sideband frequencies and onward to velocities. It will also convert them back.

Example

Example 1: ConvertFrequencyTask

```
<pre>
from herschel.hifi.generic.task import FitterTask
# Assume that htp and sds exist
cft = ConvertFrequencyTask()
cft.sds = sds            # set the HifiSpectrumDataset
cft.to = -"velocity"               # conversion to velocity
cft.reference = 1902.055688    # reference frequency in GHz
cft.inupper = True         # in the upper side band
cft()               # run!
</pre>
```

API Summary

Jython Syntax

```
see example below
```

Properties

HifiTimelineProduct **htp** [INPUT, OPTIONAL, default=null]
HifiSpectrumDataset **sds** [INPUT, OPTIONAL, default=null]
String **to** [INPUT, MANDATORY, default=null]
Double **reference** [INPUT, OPTIONAL, default=null]
Boolean **inupper** [INPUT, OPTIONAL, default=true]

API details

Properties

HifiTimelineProduct htp [INPUT, OPTIONAL, default=null]

to be processed. I.e. all SDS within the HTP are processed.

HifiSpectrumDataset sds [INPUT, OPTIONAL, default=null]

to be processed: All IF-frequencies are replaced. The MetaData items WAVENAME and WAVEUNIT are replaced. One of htp or SDS is obligatory.

String to [INPUT, MANDATORY, default=null]

Specifying where to convert to. Possible values are:

```
frequency      MHz     intermediate frequency
usbfrequency   GHz     Upper side band frequency
lsbfrequency   GHz     Lower side band frequency
velocity       Km/s    relative velocity wrt reference frequency
```

Double reference [INPUT, OPTIONAL, default=null]

The reference frequency in GHz.

Boolean inupper [INPUT, OPTIONAL, default=true]

When converting to velocity it needs to know whether it is in the upper or lower side band.

History

- 10-06-2009 DK

1.28. DataframeCount

Full Name:	herschel.hifi.pipeline.spg.plugin.DataframeCount
Alias:	DataframeCount
Type:	Jython Task -
Import:	from herschel.hifi.pipeline.spg.plugin import DataframeCount
Category:	task

Description

A table dataset of the expected and actual dataframes after an observation has been

run on the HIFI instrument. Specifically designed for AOT observations

Example

Example 1: Example of how to use DataframeCount

```
<pre>
# 268458902 ilt_fm_3 ModeIII2Test1b1fs HifiSScanModeFSwitchNoRef
# 268496255 ilt_fm_5 ModeI2RTest1c HifiMappingModeDBSRaster
# 268496331 ilt_fm_5 ModeII1Test1b HifiMappingModeOTF
# 268496316 ilt_fm_5 ModeI2Test3 HifiPointModeFastDBS
from herschel.hifi.scripts.users.share.DataframeCount import *
d = DataframeCount()(obsid=268496331, db="ilt_fm_5_prop", extratime=0)
</pre>
```

API Summary

Jython Syntax

```
DataframeCount()(obsid = 268xxxxxx, db="ilt_fm_5_prop", extra-
time=0)
```

Properties

Long **obsid** [INPUT, MANDATORY, default=no default value]
String **db** [INPUT, OPTIONAL, default=NULL]
String **host** [INPUT, OPTIONAL, default=NULL]
Integer **extratime** [INPUT, OPTIONAL, default=0]
TableDataset **results** [OUTPUT, OPTIONAL, default=NULL]

Limitations

The is a limitation on the amount of data that can be loaded into memory. Queries involving large numbers of packets can take a very long time. This will be noticed even more by remote clients. Specifically designed for AOT observations only.

API details

Properties

Long obsid [INPUT, MANDATORY, default=no default value]

obsid = 268xxxxxx

String db [INPUT, OPTIONAL, default=NULL]
Database: db = "ilt_fm_5_prop" # default provided by user properties

String host [INPUT, OPTIONAL, default=NULL]
Hostname: host = "iccdb.sron.rug.nl" # default provided by user properties

Integer extratime [INPUT, OPTIONAL, default=0]
Allows for the fact that not all HIFI tm-packets assocaited with an observation are contained between the begin and end time data stored in the tm-packets.

TableDataset results [OUTPUT, OPTIONAL, default=NULL]
TableDataset showing the number of expected vs actual dataframes per observations

See also

- reference

History

- 2007-06-01 - KE: First release
- 2008-08-15 - KE: Updated for new bbtypes used and checks if WBS / HRS Dataframes are downlinked

1.29. DatasetWrapper

Full Name:	herschel.hifi.pipeline.product.DatasetWrapper
Type:	Java Class -
Import:	from herschel.hifi.pipeline.product import DatasetWrapper
Category:	Products

Description

DatasetWrapper is a (very) simple wrapper to transform a Dataset into

a Product. The DatasetWrapper copies all MetaData of the Dataset to the MetaData of the Product and susequently makes the MetaData of the Dataset identical to that of the Product.

Example

Example 1: In Jide:

```
n/a
```

History

- 05-07-2007 DK

1.30. DisplayDataFrameTask

Full Name:	herschel.hifi.dp.access.DisplayDataFrameTask
Alias:	DisplayDataFrameTask
Type:	Java Task -
Import:	from herschel.hifi.dp.access import DisplayDataFrameTask

Description

DP Access module: DisplayDataFrameTask

Displays DataFrame objects. There are no arguments assocaited with this task at the current time.

Example

Example 1: DisplayDataFrameTask

```
<pre>
# Usage from jython environment would be:
from herschel.hifi.dp.access import DisplayDataFrameTask
DisplayDataFrameTask()(your_DataFrame_object)
#The data frame passed to the program can be
#Wbs or Hrs Hifi data frame.
</pre>
```

1.31. DisplayPacketTask

Full Name:	herschel.hifi.dp.access.DisplayPacketTask
Alias:	DisplayPacketTask
Type:	Jython Task -
Import:	from herschel.hifi.dp.access import DisplayPacketTask
Category:	task

Description

DisplayPacketTask is a task used to derive parameter values from an array of

housekeeping packets. The DisplayPacketTask can be called on the command line as well as via a GUI. The GUI facility allows the user to select a parameter and to view the associated values via a plot window. The user can export the selected parameter to an ascii and/or a fits file from the GUI as well. The 'raw' values do not contain any information about the units of the parameter. The 'converted' values are scaled using the units stored in the database. Not all 'raw' house keeping parameters will have 'converted' values. In the command line mode the user may call this task with the pname='ALL'. This will create a complete TableDataset of all parameter values from all housekeeping packets. Using pname = 'ALL' is a time comsuming process.

Example

Example 1: How to use DisplayPacketTask

```
<pre>
from herschel.hifi.dp.access import DisplayPacketTask
from herschel.hifi.dp.access import AccessPacketTask
# interactive mode (GUI):
hk = AccessPacketTask()(gui=1)
hktask = DisplayPacketTask()
hktask.gui=1
hktask.hkarray=hk
# for each param selected in the GUI a TableDataset is created
paramTblDataset = hktask.result
# non-interactive mode (Database: ilt_fm_5_prop):
hk = AccessPacketTask()(obsid=268510862)
# a specific parameter pname can be passed
result = DisplayPacketTask()(gui=1, hkarray=hk, pname='BB_ID')
# or all parameters can be requested
result = DisplayPacketTask()(gui=1, hkarray=hk, pname='ALL')
</pre>
```

API Summary

Jython Syntax

```
Interactive mode:
table = DisplayPacketTask()(gui=1, hkarray=<myarray>)
Non-iteractive mode:
table = DisplayPacketTask()(hkarray=<myarray>, pname=<my-pname>)
```

Properties

array **hkarray** [INPUT, MANDATORY, default=None]
String **pname** [INPUT, OPTIONAL, default=None]

Properties
Boolean **convert** [INPUT, OPTIONAL, default=false]
Boolean **gui** [INPUT, OPTIONAL, default=false]
TableDataset **result** [OUTPUT, OPTIONAL, default=None]

Limitations

The is a limitation on the amount of data that can be loaded into memory.

Miscellaneous

The hk-array cannot be selected from within this GUI

API details

Properties

array hkarray [INPUT, MANDATORY, default=None]

Input array containing objects of the type TmSourcePacket

String pname [INPUT, OPTIONAL, default=None]

Allows user to select a specific type of TmSourcePacket from the hkarray. The pname = 'ALL' is only available from the command line.

Boolean convert [INPUT, OPTIONAL, default=false]

When true it passes the converted (TmSourcePacket with units) values as well, if there is no conversion factor nothing is plotted in the GUI

Boolean gui [INPUT, OPTIONAL, default=false]

When gui = true a GUI will be provided to the user Note: This option is currently forced true

TableDataset result [OUTPUT, OPTIONAL, default=None]

Creates a four column TableDataset. The first column is the raw relative TmSourcePacket times, second is the relative converted TmSourcePacket times, third is the raw selected parameter and the fourth is the converted (parameter with units) column. In case pname = 'ALL', all pnames will be included in the table (Expensive in calculation time, but possible).

See also

- reference

History

- 2005-09-09 - PZ: First release
- 2006-05-22 - KE: Last Update: Fixed documentation format
- 2006-06-07 - PZ: adopt to Task.views API
- 2006-08-08 - KE: Updated this task with gui plot
- 2007-02-09 - KE: Added buttons to export the TableDataset to a file (ASCII / FITS)

- 2007-06-29 - KE: Removing gui components from constructor
- 2008-04-16 - KE: Updated due to changes in the Plot package
- 2009-04-15 - KE: Updated to reflect changes in IA_TASK - TaskParameter Flushing

1.32. DoAntennaTemp

Full Name:	herschel.hifi.pipeline.generic.DoAntennaTemp
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoAntennaTemp

Description

Translates to the antenna temperature scale.

Example

Example 1: From jide:

```
doAntennaTemp(htp=htp, cal=my_forw_eff)
doAntennaTemp(htp=htp, forwardEff=0.68)
doAntennaTemp(htp=htp, calibration=obs.calibration)
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

GenericPipelineCalProduct **cal** [INPUT, OPTIONAL, default=no default value]

MapContext **calibration** [INPUT, OPTIONAL, default=no default value]

Double **forwardEff** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

GenericPipelineCalProduct cal [INPUT, OPTIONAL, default=no default value]

A calibration product that provides the forward efficiencies. The product allows for LO-specific parameters.

MapContext calibration [INPUT, OPTIONAL, default=no default value]

Parameter to pass a calibration context from which all the calibration input, here the forward efficiencies, can be retrieved.

Double forwardEff [INPUT, OPTIONAL, default=no default value]
The forward efficiency to apply.

History

- 2009-03-28 - melchior: : new implementation.
- 2009-08-27 - melchior: : Update for new calibration product structure.
- 2010-02-01 - melchior: : Allow to pass calibration context.

1.33. DoAverage

Full Name:	herschel.hifi.pipeline.generic.DoAverage
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoAverage

Description

Iterates through all the datasets and takes the average for each group of datasets which are defined by the type ("sds_type") and frequency group.

Note that for the proper execution of this module the CheckFreqGrid-task should be run beforehand so that the frequency groups are available (and added as meta data to the datasets). If these groups are not avaliable all the datasets for a given type are averaged. Most of the work is delegated to the task herschel.ia.toolbox.spectrum.AverageSpectrum. In particular, the

- the different processing modes that define how flags and weights should be included in the processing can be set by the "variant" parameter;
- the other attribute data (other data columns in the table datasets) are processed according to the rules defined for HIFI in the spectrum toolbox.

Note that this task requires the output to be assigned to a variable (in the form `htp=doAvg(...)`). The type of the output depends on the value set for `return_single_ds`.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoAverage
htp = doAvg(htp=htp, params=params) # the standard pipeline mode
htp = doAvg(htp=htp, return_single_ds=False) # the standard pipeline mode
# the following returns a dataset with just science data. Note that the
grouping
# treats the ON and OFF scans in different groups so these are not mixed.
doAvg(htp=htp, selection_meta=["science", -"hc"], return_single_ds=True)
doAvg(htp=htp, selection_meta=["scienceOff"], return_single_ds=True)
doAvg(htp=htp, selection_meta={"frequencyGroup":["1","2"]},
return_single_ds=True)
doAvg(htp=htp, selection={"bbtype":[6005,6031]}, return_single_ds=True)
ds = doAvg.result
# The following averages over all the (selected) scans in the htp --
irrespective of a grouping
# The result typically is a dataset with a single row.
doAvg(htp=htp, selection={"bbtype":[6005,6031]}, return_single_ds=True,
preserveGroups=False)
ds = doAvg.result
```

API Summary

Jython Syntax

```
See example below.
```

Properties

```
HifiTimelineProduct htp [IN, MANDATORY, default=no default value]
Object result [OUT, MANDATORY, default=no default value]
```

Properties
Boolean **ignore** [INPUT, OPTIONAL, default=no default value]
Object **selection_meta** [INPUT, OPTIONAL, default=no default value]
Object **selection** [INPUT, OPTIONAL, default=no default value]
Boolean **preserveGroups** [INPUT, OPTIONAL, default=no default value]
String **variant** [INPUT, OPTIONAL, default="flux-weight".]
Boolean **return_single_ds** [INPUT, OPTIONAL, default=True]
Double **loThrow_tolerance** [INPUT, OPTIONAL, default=no default]

API details

Properties

HifiTimelineProduct htp [IN, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

Object result [OUT, MANDATORY, default=no default value]

The result of the average operation.

Boolean ignore [INPUT, OPTIONAL, default=no default value]

Flag to indicate whether the execution of the module should be ignored.

Object selection_meta [INPUT, OPTIONAL, default=no default value]

Allows to specify a selection of datasets included in the timeline product by just looking at specific meta data values. Basically, there are two possibilities:

- As a py dictionary with keys specifying the meta data key and values the admissible values. These values are specified as lists.
- As a list of admissible 'sds_type's (the type appearing in the summary table). Hence this is internally translated into a py dictionary with 'sds_type' set as key. In order to distinguish ON and OFF science datasets, wee (artificially) allow for the values "scienceOn" and "scienceOff".

Object selection [INPUT, OPTIONAL, default=no default value]

This selection allows to make sub-selection from the set of all scans by looking up matches for suitable values to be found in specific columns of the dataset. Here, the functionality of the the AverageSpectrum-task ('avg') sitting in the spectrum toolbox (ia.toolbox.spectrum) is reused. See there for further details.

Boolean preserveGroups [INPUT, OPTIONAL, default=no default value]

Specifies that the groups should be preserved in the sense that datasets that are associated with different groups are not mixed. Note that the grouping is a concept defined on the level of the meta data of the datasets. This grouping is also used when cleaning up data (see doCleanUp for further information). In case some of the meta data needed to specify the grouping is missing the average is perfomed on a per dataset basis.

String variant [INPUT, OPTIONAL, default="flux-weight".]

Specify the variant of processing mode you would like to run (including flags / weights / ...). Possible values are "flux" / "flux-wave" / "flux-wave-weight" / "flux-flag-wave" / "flux-weight-flag-wave"

- "flux": average flux, wave set to wave of the first input spectrum, add weights, propagate flags with OR.
- "flux-wave": average flux, average wave, add weights, propagate flags with OR.
- "flux-weight-wave": weighted average flux, weighted average wave, add weights, propagate flags with OR.
- "flux-flag-wave": filter for non-flagged and average flux, filter for non-flagged and average wave, filter for non-flagged and add weights, set flag to 0; in case there are no unflagged values, straight average of flux and weights and add all weights but set the flag to a value propagated with OR.
- "flux-weight-flag-wave": the same as flux-flag-wave but using weighted averages.

Boolean return_single_ds [INPUT, OPTIONAL, default=True]

Return a single dataset with the average or, in case you have several groups to handle in the timeine product, the averages. Each line in the result will correspond to one of the groups. You can identify the groups from the bbtype, LO Frequency, rasterLineNum/rasterColumnNum, or scanLineNum. Note that, typically, this does not allow you to include comb datsets to be included in the average. Be aware that if you set this option to False the original timeline product will be overwritten. This is the default pipeline setting since you want to have a timeline product back.

Double loThrow_tolerance [INPUT, OPTIONAL, default=no default]

The groups to be averaged are checked for having consistent LO throw values. These values are said to be consistent if the LO throw changes in the given group are smaller than a given tolerance. The tolerance is computed from the parameter specified here by multiplying the value with the average LO frequency. The value specified here is specified in units km/sec.

History

- `2008-04-04 - meli`: initial

1.34. DoBadLo

Full Name:	herschel.hifi.pipeline.level0.DoBadLo
Type:	Java Task -
Import:	from herschel.hifi.pipeline.level0 import DoBadLo

Description

Task to flag in a HifiTimelineProduct the bands in the spectra that have a bad LO values due foreseen Spurs,

instabilities , saturations or for any generic reason the LO is marked as Bad. The Task expect that the "cal" input Product will contain a TableDatset with the Columns: {@link #BAD_LO_NAME}, {@link #BAD_LO_WIDTH_NAME}, {@link #BAD_LO_BANDS_NAME}, {@link #BAD_LO_TYPE_NAME}.

The {@link #BAD_LO_NAME} contains the central frequency where the LO is not good. Repeated value is possible if the same frequency has more then one type of problems.

The {@link #BAD_LO_WIDTH_NAME} contains the width of the range of bad frequencies.

The {@link #BAD_LO_TYPE_NAME} contains one type of problem at the specified frequencies.

The {@link #BAD_LO_BANDS_NAME} contains the bands that have the problem at that specified frequency.

A new Column with name {@link #BAD_LO_FLAG_NAME} is added to all the HifiSpectrum-Dataset contained in the HifiTimelineProduct. For each spectrum it contains a set of values (one for each subband) correspondent to the flag that have been raised for that band. The value is equivalent to a binary number where each bit is a flag. The correspondence between flag and bit is the flag position in the FLAG_NAMES vector.

If some bands in the HifiSpectrumDataset are flagged, a StringParameter containing all the flag raised in the specific subband is added to the MetaData of the HifiSpectrumDataset. The String Parameter name is {@link #META_FLAG_RADIX}Nwhere N is the specific subband.

Example

Example 1: From jide/hype:

```
spurs=
obs.calibration.getCalNode("downlink").getCalNode("generic").getProduct("spurs");
  DoBadLo(htp=htp, cal=spurs)
```

API Summary

Jython Syntax

```
See example below.
```

Properties

```
HifiTimelineProduct htp [IO, MANDATORY, default=No default value]
Product cal [INPUT, MANDATORY, default=No default value]
```

API details

Properties

HifiTimelineProduct htp [IO, MANDATORY, default=No default value]

The HifiTimelineProduct which will be updated with the flag

Product cal [INPUT, MANDATORY, default=No default value]

Contains a TableDatset dataset which contains the Columns:

{@link #BAD_LO_NAME}, {@link #BAD_LO_WIDTH_NAME}, {@link #BAD_LO_BANDS_NAME}, {@link #BAD_LO_TYPE_NAME}

1.35. DoChannelWeights

Full Name:	herschel.hifi.pipeline.generic.DoChannelWeights
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoChannelWeights

Description

Computes the (channel-dependent) weights and fills them into the datasets of type 'science'.

Different options are available:

- Radiometric formula (definition = "radiometric").
- Integration time only (definition = "integrTime").
- Sample variance obtained from a moving window in the flux data (definition = "variance").

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoChannelWeights
doWeights = DoChannelWeights()
doWeights(htp=htp, definition="radiometric", interp="linear"))
```

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default Value]
CalFluxHotCold **cal** [INPUT, OPTIONAL, default=no default value]
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]
String **definition** [INPUT, OPTIONAL, default=no default value]
String **smoothing** [INPUT, OPTIONAL, default=no default value]
Integer **width** [INPUT, OPTIONAL, default=no default value]
String **interpolator** [INPUT, OPTIONAL, default=no default value]
Boolean **ignore** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default Value]

The timeline product (observation) to be passed to the module.

CalFluxHotCold cal [INPUT, OPTIONAL, default=no default value]

The product containing the sysytem temperature and the band pass of the black body calibrators as obtained in the MkFluxHotCold-module.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Configuration parameter that can be passed to the product

String definition [INPUT, OPTIONAL, default=no default value]

Definition of the weights to be used: 'radiometric', 'tsys', 'integrTime', 'variance'

String smoothing [INPUT, OPTIONAL, default=no default value]

The smoothing operation applied in case a width > 1 is specified. By default a Gaussian filter is used.

Integer width [INPUT, OPTIONAL, default=no default value]

The window width used when computing the (local) sample variance or when smoothing the radiometric formula output.

String interpolator [INPUT, OPTIONAL, default=no default value]

The interpolator used when interpolating the system temperature.

Boolean ignore [INPUT, OPTIONAL, default=no default value]

Flag to indicate whether the execution of the module should be ignored.

History

- `2007-12-18 - meli`: Initial version.
- `2008-05-08 - meli`: Dependency on ScanInterpolator removed

1.36. DoCleanUp

Full Name:	herschel.hifi.pipeline.generic.DoCleanUp
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoCleanUp

Description

Module to remove data that is no longer needed after running the generic pipeline.

Furthermore, it packs all the data belonging to the same frequency group to on single dataset. Here, the temporal order is conserved.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoCleanUp
cleanUp(htp=htp) # apply just the defaults
params = herschel.hifi.pipeline.generic.PipelineConfiguration.getConfig(htp)
cleanUp(htp=htp, params=params) # apply the defaults set for the pipeline
cleanUp(htp=htp, mergeDatasets=False) # do not merge the datasets belonging
to the same group
cleanUp(htp=htp, retain='science') # retain all the science data in the
timeline product, ON and OFF's.
```

API Summary

Jython Syntax

See example below.

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]

String **retain** [INPUT, OPTIONAL, default=no default value]

Boolean **mergeDatasets** [INPUT, OPTIONAL, default=True]

Integer **datasetSize** [INPUT, OPTIONAL, default=100]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Configuration parameter that can be passed to the product

String retain [INPUT, OPTIONAL, default=no default value]

String parameter specifying which data to retain while cleaning up. Options: 'scienceOn' (default), 'science'.

Boolean mergeDatasets [INPUT, OPTIONAL, default=True]

Flag that can be set for whether the datasets that belong to the same "group" should be merged or not. The data typically is structured into groups where the groups are given an observing mode specific meaning:

- For spectral scans, all the scans with the same LO tuning form a group.
- For raster maps, all the scans that refer to the same raster point ("rasterColumnNum", "rasterLineNum") form a group.
- For otf maps, all the scans in the same line ("scanLineNum") form a group.
- For cross maps, all the scans associated with the same point within the cross ("customMapPointNum") form a group.

In addition, scans that belong to the same group but are not subsequent in observation time will not be merged.

Integer datasetSize [INPUT, OPTIONAL, default=100]

Allows to specify a maximum size of the datasets to be included in the result timeline product. Note that datasets are not teared in the sense that datasets are not merged if the resulting size is not within the specified size. Furthermore, if a given input dataset is already above the limit this will be moved to the output without change.

History

- `2008-04-04 - meli`: initial
- `2008-07-17 - meli`: Parameters mergeDatasets and params added. Some refactoring to optimize memory.

1.37. DoDeconvolution

Full Name:	herschel.hifi.dp.deconvolution.DoDeconvolution
Type:	Java Task -
Import:	from herschel.hifi.dp.deconvolution import DoDeconvolution

Description

The deconvolution tool is a post-Level 2 processor to separate the "folded" double sideband (DSB) data inherently produced

by the heterodyne process into a single sideband (SSB) result.

Example

Example 1: Description

```
from herschel.ia.task import Task
from herschel.ia.task import TaskParameter
from herschel.ia.obs import ObservationContext
from herschel.hifi.pipeline.product import HifiTimelineProduct
from herschel.hifi.dp.deconvolution import DoDeconvolution
#Do deconvolution with default input values.
decon_result = doDeconvolution(obs=my_obs)
#Do deconvolution with user specified input values.
decon_result =
doDeconvolution(obs=my_obs,polarization=0,max_iterations=200,tolerance=0.0010,channel_weighting
```

API Summary

Jython Syntax

```
decon_result = doDeconvolution(obs=my_obs)
```

Properties

ObservationContext **obs** [INPUT, MANDATORY, default=no default value.]
Integer **polarization** [INPUT, OPTIONAL, default="H_POL"]
Integer **max_iterations** [INPUT, OPTIONAL, default="200"]
Double **tolerance** [INPUT, OPTIONAL, default="0.001"]
Integer **gain** [INPUT, OPTIONAL, default="NO_GAIN"]
Boolean **channel_weighting** [INPUT, OPTIONAL, default="false"]
Integer **parameter_ignore_mask** [INPUT, OPTIONAL, default="524288"]
Integer **plot_dsb** [INPUT, OPTIONAL, default="NO_PLOT"]
Double **lambda1_channels** [INPUT, OPTIONAL, default="0.0"]
Double **lambda2_gains** [INPUT, OPTIONAL, default="0.0"]
Double **cont_offset** [INPUT, OPTIONAL, default="0.0"]
Boolean **expert** [INPUT, OPTIONAL, default="false"]
Product **decon_result** [OUTPUT, MANDATORY, default=no default value]
Product **interim_output** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

ObservationContext obs [INPUT, MANDATORY, default=no default value.]

- ObservationContext from Level 2 Pipeline, (containing htpUpper, htpLower)

Integer polarization [INPUT, OPTIONAL, default="H_POL"]

- Polarization. Two possible choices, H_POL(=0), V_POL(=1).

Integer max_iterations [INPUT, OPTIONAL, default="200"]

- Maximum number of iterations.

Double tolerance [INPUT, OPTIONAL, default="0.001"]

- Convergence tolerance epsilon - when old_chi^2-_chi^2 < epsilon, stop. Default=0.001.

Integer gain [INPUT, OPTIONAL, default="NO_GAIN"]

- Upper and lower side bands gains.

Boolean channel_weighting [INPUT, OPTIONAL, default="false"]

- DSB weights channel by channel

Integer parameter ignore_mask [INPUT, OPTIONAL, default="524288"]

- Flag to check if the scan is bad that should be skipped.

Integer plot_dsb [INPUT, OPTIONAL, default="NO_PLOT"]

- Plot options, including ssb, usb and lsb.

Double lambda1_channels [INPUT, OPTIONAL, default="0.0"]

- SSB entropy weight in the chisquare() function minimization.

Double lambda2_gains [INPUT, OPTIONAL, default="0.0"]

- Gain entropy weight in the chisquare() function minimization.

Double cont_offset [INPUT, OPTIONAL, default="0.0"]

- Continuum offset needed for model.

Boolean expert [INPUT, OPTIONAL, default="false"]

- Expert mode for generating interim output.

Product decon_result [OUTPUT, MANDATORY, default=no default value]

- Product of doDeconvolution task. Output. It contains the following: ssb - Single side band frequency and flux. gain - LO frequency, USB_gain, and LSB_gain. redundancy - freq_bins and dsb_hist.

Product interim_output [OUTPUT, MANDATORY, default=no default value]
- Interim product of doDeconvolution task. Output. It contains the following: dsbSpectrum, interim_lssb, interim_lgain, interim_chiSquare.

1.38. DoFluxHotCold

Full Name:	herschel.hifi.pipeline.generic.DoFluxHotCold
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoFluxHotCold

Description

This module applies the bandpass spectra (typically obtained

from the MkFluxHotCold module) to the science spectra for the intensity calibration.

The bandpass spectra are passed to the module by using the keyword 'cal' and should be of type {@link CalFluxHotCold}. This product contains both, bandpass spectra and receiver temperature - for this module, only the bandpass is needed.

Typically, several bandpass spectra are included in the calibration product - each associated with different observation times and/or LO frequencies. For each science spectrum to be calibrated a suitable bandpass spectrum is constructed by first picking the set of bandpass spectra with consistent LO frequency and then applying a suitable interpolation scheme so that a bandpass with consistent observation time is obtained.

A validation mechanism checks the spectra to be subtracted from each other and, if needed, sets in the result data a row flag. The default mechanism inspects the mixer currents (columns 'MJC_hor' or 'MJC_Ver', respectively) and tests their relative deviation (of the science data and the bandpass) to be less than a given tolerance. Once the tolerance is exceeded, for the result spectrum a row flag (bit 18) is set. The tolerance can be set by specifying the 'validatorTolerance'-parameter. By default it is set to 0.025. This default checking mechanism can be overruled by passing as 'validator'-parameter an instance of the {@link DataValidator} interface. When setting this parameter to None, no checking is done.

Examples

Example 1: From jide:

```
from herschel.hifi.pipeline.product import CalFluxHotCold
# htp is a HifiTimelineProduct with spectra to be calibrated.
# calHotCold is a CalFluxHotCold
res = doFluxHotCold(htp=htp, cal=calHotCold)
res = doFluxHotCold(htp=htp, cal=calHotCold, interpolator="LINEAR")
```

Example 2: From jide:

```
from herschel.hifi.pipeline.product import CalFluxHotCold
from herschel.hifi.cal import  CalCoupCoeff
htp is a HifiTimelineProduct with  spectra to be calibrated.
coupl are coupling coefficients (CalCoupCoeff)
coupl=CalCoupCoeff()
cal=mkFluxHotCold(htp=htp, coeff=coupl)
res = doFluxHotCold(htp=htp, cal=cal)
```

Example 3: From jide:

```
# the validator tolerance is re-set.
htp = doFluxHotCold(htp=htp, cal=cal, validatorTolerance=0.1)
```

Example 4: From jide:

```
# validation is switched off.
```

Example 4: From jide:

```
htp = doFluxHotCold(htp=htp, cal=cal, validator=None)
```

Example 5: From jide:

```
# an alternative validation mechanism is set (here based, as dummy, on the
LoFrequency)
from herschel.hifi.pipeline.generic.dorefsubtract import
SubtractionDataValidatorImpl
validator = SubtractionDataValidatorImpl()
validator.configure('LoFrequency', 0.001, True)
htp = doFluxHotCold(htp=htp, cal=cal, validator=validator)
```

API Summary

Jython Syntax
See example below

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
CalFluxHotCold **cal** [INPUT, MANDATORY, default=no default value]
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]
Object **validatorTolerance** [INPUT, OPTIONAL, default=0.025]
DataValidator **validator** [INPUT, OPTIONAL, default=no default value]
MapContext **calibration** [INPUT, OPTIONAL, default=no default value]
String **interpolator** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]

The timeline product (observation) to be processed.

CalFluxHotCold cal [INPUT, MANDATORY, default=no default value]

Provides the product with the bandpass and receiver temperature spectra.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Object containing pipeline configuration parameters (such as interpolation scheme to be used).

Object validatorTolerance [INPUT, OPTIONAL, default=0.025]

The tolerance to be used for the default validator that checks the difference in the mixer currents and sets a row flag once the mixer currents found in the science data and the bandpass deviate by more than the tolerance from each other. A relative distance measure is used to quantify the deviation (`2abs(x1-x2)/(abs(x1)+abs(x2))`). Alternatively, a calibration ob-

Object validatorTolerance [INPUT, OPTIONAL, default=0.025]

ject of type `GenericPipelineCalProduct`can be passed from which the band-specific tolerances will be extracted. Yet another possibility is to pass the whole calibration context - the task retrieves the appropriate product with the tolerances. Furthermore, you can use the parameter 'calibration' for that purpose. Once a product of type `GenericPipelineCalProduct` is obtained, the appropriate tolerance is looked up from a table (associated with the given date) from a column named 'doFluxHotCold'.

DataValidator validator [INPUT, OPTIONAL, default=no default value]

Custom validator that can be passed to the task to check source and ref that is subtracted from each other and flag data once the tolerance exceeds a given tolerance. Note that the validatorTolerance cannot be used to specify this threshold. Furthermore, the default validator is overwritten once a validator or None is passed here.

MapContext calibration [INPUT, OPTIONAL, default=no default value]

Parameter to pass a calibration context from which all the calibration input, here just mixer current tolerances, can be retrieved.

String interpolator [INPUT, OPTIONAL, default=no default value]

Interpolation scheme to be used. Currently it accepts the following options:

- "PREVIOUS" the previous hot-cold measurement in time is used.
- "NEXT" the next hot-cold measurement in time is used.
- "LINEAR": a linear interpolation is used.
- "NEAREST": a nearest neighbor interpolation is used.
- "CUBIC_SPLINE" : a polynomial cubic Spline fit is done.
- "TRUNC_LINEAR" : linear for bracketed scans, nearest neighbour scans with only a single neighbouring scan.

History

- `2009-09-24 - Melchior`: Validators included.

1.39. DoFold

Full Name:	herschel.hifi.pipeline.generic.DoFold
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoFold

Description

Performs folding for frequency switched spectra.

The folded spectra are constructed by averaging the original spectra with a shifted and inverted copy. This algorithm corresponds to the most simple scheme described in the article {\it "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.

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.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoFold
doFold = DoFold()
doFold(htp=htp)
doFold(htp=htp, throw=0.120, shift=True)
doFold(htp=htp, throw=120, unit="MHz", shift=True)
doFold(htp=htp, params=params)
doFold(htp=htp, throw=120, unit="MHz", shift=True, types="science,hc")
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

Properties
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]
Double **throw** [IN, OPTIONAL, default=no default value]
String **unit** [INPUT, OPTIONAL, default=no default value.]
Boolean **shift** [IN, OPTIONAL, default=False]
String **types** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product to be processed.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Configuration parameter that can be passed to the product.

Double throw [IN, OPTIONAL, default=no default value]

The frequency throw to be assumed. If now throw is specified the meta data field 'loThrow' of each dataset is used (if available).

String unit [INPUT, OPTIONAL, default=no default value.]

Specify the unit the frequency throw is expressed in - in case it is explicitly specified. If no unit is specified it is assumed that the unit of the frequency throw is the same as the unit of the wave scale. Typical values are "GHz" or "MHz".

Boolean shift [IN, OPTIONAL, default=False]

Shifts the folded spectra by half the throw in direction of the throw.

String types [INPUT, OPTIONAL, default=no default value]

Restrict the folding to the specified sds_types. A comma-separated list of values can be entered here.

History

- `2009-05-15 - meli`: Initial implementation.

1.40. DoFreqGrid

Full Name:	herschel.hifi.pipeline.generic.DoFreqGrid
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoFreqGrid

Description

Computes a frequency scale which will be used for resampling

and resamples the flux data to this grid. Optionally, frequency grids may be specified by the users.

Example

Example 1: from jide:

```
doFreqGrid(htp=htp, resolution=1.0)
newGrid = mkFreqGrid(htp, stepsize=0.5)
doFreqGrid(htp=htp, grid=newGrid)
```

API Summary

Jython Syntax
`See example below.`

Properties
`HifiTimelineProduct` **`htp`** `[INOUT, MANDATORY, default=no default value]`
`PipelineConfiguration` **`params`** `[INPUT, OPTIONAL, default=no default value]`
`String` **`scheme`** `[INPUT, OPTIONAL, default=no default value.]`
`Double` **`resolution`** `[INPUT, OPTIONAL, default=no default value]`
`Unit` **`unit`** `[INPUT, OPTIONAL, default=no default value]`
`Object` **`grid`** `[INPUT, OPTIONAL, default=no default value]`

API details

Properties

`HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]`

The timeline product to be processed.

`PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]`

Configuration parameter that can be passed to the product

`String scheme [INPUT, OPTIONAL, default=no default value.]`

The scheme to be adopted for the resampling - available are "trapezoidal" or "euler".

Double resolution [INPUT, OPTIONAL, default=no default value]

The desired resolution of the output frequency grid (in MHZ).

Unit unit [INPUT, OPTIONAL, default=no default value]

The unit the desired resolution of the output frequency grid is specified in. By default, it is specified in the same unit as the frequency scale of the spectra to be resampled.

Object grid [INPUT, OPTIONAL, default=no default value]

Some specification of (a) grid(s):

- as PyDictionary: For each dataset a frequency grid is specified. Use the integer key of the dataset within the product as key in the PyDictionary and a Double1d[] as values.
- as CalFreqGrid: Calibration product that is created in the MkFreqGrid module.

1.41. DoGridding

Full Name:	herschel.hifi.dp.otf.DoGridding
Type:	Java Task -
Import:	from herschel.hifi.dp.otf import DoGridding
Category:	HIFI

Description

produces cube(s) of images from a HifiTimelineProduct.

This task makes a cube for each specified subband (segment) of the HifiTimelineProduct.

If no subband is specified, it will make an array of cubes, containing one cube per subband present in the science datasets within the HifiTimelineProduct.

The images are produced by performing two dimensional gridding of a number of spectra irregularly distributed over some area of the sky observed in a mapping mode. The spectra are read from a number of HifiSpectrumDatasets within the HifiTimelineProduct.

This task computes a regular grid that covers the region of the sky where the spectra have been observed.

For each pixel of that grid, the task computes a normalized convolution of those spectra which fall in the convolution kernel around such pixel.

The user can choose between assigning the same weight to every spectrum or using the weights computed by the generic part of the HIFI pipeline (i.e. use the weights columns of the HifiSpectrumDatasets).

Besides, the user has to choose between several type of filter functions to compute the contribution of each spectrum to each pixel, based on the distance of the spectrum to the centre of the kernel.

The user can also use an smoothing factor and provide some other parameters, e.g. to control the geometry of the regular grid and the range of channels to use to create the cube (i.e. to clip the wave range).

Examples

Example 1: make cube for first subband

```
subband = 1
cubes = doGridding(htp=htp, subbands=Int1d([subband]))
cube = cubes[0]
```

Example 2: make cubes for all the subbands, then Display the first one

```
cubes = doGridding(htp=htp)
cubes_count = len(cubes)
cube = cubes[0]
Display(cube)
```

Example 3: select some datasets instead of picking all the "science" datasets

```
cubes = doGridding(htp=htp, subbands=Int1d([2,4]), datasetIndices=([3,4,5]))
cubes = doGridding(htp=htp, subbands=Int1d([2,4]),
dataset_indices=Int1d([3,4,5]))
cube_subband_2 = cubes[0]
cube_subband_4 = cubes[1]
```

Example 4: select some datasets and specify beam dimensions

```
cubes = doGridding(htp=htp, subbands=Int1d([2,4]),
datasetIndices=Int1d([5,6,7,9]), beamSize=Double1d([20., 40.]) -)
cube_subband_2 = cubes[0]
cube_subband_4 = cubes[1]
```

Example 5: make the cube only for the subbands 2 and 4:

```
cubes = doGridding(htp=htp, subbands=Int1d([2,4]),
datasetIndices=Int1d([5,6,7,9]), beamSize=Double1d([20., 40.]) -)
```

API Summary

Jython Syntax

```
cubes = doGridding(htp=htp [, subbands=Int1d([i,j, ..., n]), ...])
cubes = doGridding(htp=htp)
```

Properties
HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default value]
Int1d **subbands** [INPUT, OPTIONAL, default=no default value]
Double **beam** [INPUT, OPTIONAL, default=no default value]
String **weightMode** [INPUT, OPTIONAL, default="equal"]
Int2d **channels** [INPUT, OPTIONAL, default=empty by default]
Int1d **mapSize** [INPUT, OPTIONAL, default=Int1d(2,0) i.e. autocompute optimal]
Double1d **refPixel** [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]
Double1d **refPixelCoordinates** [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]
Double1d **pixelSize** [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]
Double1d **smoothFactor** [INPUT, OPTIONAL, default=Double1d(2, 1.0), i.e. autocompute]
String **filterType** [INPUT, OPTIONAL, default="gaussian"]
Double1d[] **filterParams** [INPUT, OPTIONAL, default=no default value]
Double1d **xFilterParams** [INPUT, OPTIONAL, default=no default value]
Double1d **yFilterParams** [INPUT, OPTIONAL, default=no default value]
String1d **datasetTypes** [INPUT, OPTIONAL, default=String1d(["science"])]
String **datasetType** [INPUT, OPTIONAL, default="science"]
Int1d **datasetIndices** [INPUT, OPTIONAL, default=no default value]
Boolean **ignoreOffs** [INPUT, OPTIONAL, default=true]
Integer **apid** [INPUT, OPTIONAL, default=no default value]
PointingProduct **pointing** [INPUT, OPTIONAL, default=no default value]

Properties
SiamProduct **siam** [INPUT, OPTIONAL, default=no default value]
TableDataset **offsetsTable** [INPUT, OPTIONAL, default=no default value]
Cube **cube** [OUTPUT, OPTIONAL, default=no default value]
Cube[] **cubes** [OUTPUT, OPTIONAL, default=no default value]
Double1d **xPoints** [OUTPUT, OPTIONAL, default=no default value]
Double1d **yPoints** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default value]

container of the spectra to be gridded to produce a cube of images.

Int1d subbands [INPUT, OPTIONAL, default=no default value]

specifies the subbands to be used, to produce a cube for each given subband of the input HifiTimelineProduct.

necessary only when the spectra in the given HifiTimelineProduct have several subbands.

Please note that in the general "spectral jargon" the subbands are often called "segments" since these subbands are not necessarily consecutive (e.g. HRS spectra).

Double beam [INPUT, OPTIONAL, default=no default value]

No default value.

Provides the antenna beam width (HPBW), **in seconds of arc**.

String weightMode [INPUT, OPTIONAL, default="equal"]

specifies how to assign spectral weights;

every channel of every spectrum receives its own weight.

The weighting strategies are:

- "equal": all channels of all spectra have the same weight.
- "selection": takes the "weight" column of the spectra.

Please note that for each spectrum, each channel can have a different weight so the weighting is done channel-wise.

Int2d channels [INPUT, OPTIONAL, default=empty by default]

by default, in this case all channels are selected for each segment (subband)

This parameter can be used to "crop" a part of spectral range.

Int2d channels [INPUT, OPTIONAL, default=empty by default]

For each segment, this Int2d has a row with two elements, the first element the start channel, and the second element specifies the last channel, i.e. the start and the end of the range of channels to be read per subband

Int1d mapSize [INPUT, OPTIONAL, default=Int1d(2,0) i.e. autocompute optimal]

this parameter may be used to specify the size of the map to be generated; the size is given along the x and y axes, in **pixels**.

When both sizes are equal, a Double1d with a single element can be provided.

Otherwise, this parameter array must contain two elements:

- width of the map, size along the x axis
- height of the map, size along the y axis

By default, this parameter is set to zero, in order to let algorithm choose the optimal map dimensions.

Double1d refPixel [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

specifies which is the reference pixel (in both axes). The coordinates of this reference pixel can also be specified by means of another optional parameter, refPixelCoordinates.

By default, this parameter is set to NaN, to let the algorithm choose an appropriate reference pixel.

If both x- and y-axis values of the reference pixel are equal, a Double1d array with a single element can be provided.

Otherwise, this parameter must have two elements:

- first element of the Double1d array specifies the value of the reference pixel in the x axis,
- second element of the Double1d array specifies the value of the reference pixel in the y axis

This reference pixel is referred to the bottom most, left most pixel of the regular grid that covers the whole area to be mapped. That is, the (0,0) pixel corresponds to the center of bottom-most, left-most pixel of the regular grid computed for the map.

Double1d refPixelCoordinates [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

specifies the coordinates of the reference point, in the x and y axes, in the same units of the input data (for HifiSpectrumDataset, usually degrees).

By default, this parameter is set to zero to let algorithm compute the offset that corresponds to the reference pixel.

Otherwise, this parameter must have two elements:

Double1d refPixelCoordinates [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

- first element: coordinate of the reference pixel in the x axis,
- second element: coordinate of the reference pixel in the y axis

Double1d pixelSize [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

defines the size of the pixels in the x and y axes, measured in arc seconds per pixel.

By default this parameter is set to zero to let algorithm compute the optimal size.

If both x- and y-axis values are equal, a Double1d array with a single parameter can be chosen.

Otherwise, this parameter must have two elements:

- size along the x-axis in the first element,
- size along the y-axis in the second one.

Double1d smoothFactor [INPUT, OPTIONAL, default=Double1d(2, 1.0), i.e. autocompute]

deprecated parameter (usage not recommended);

provides a smooth factor, increasing (broadening) the area of influence (kernel size) used to compute each pixel.

String filterType [INPUT, OPTIONAL, default="gaussian"]

specifies the type of convolution filter, by default an exponential (gaussian) function.

Allowed values:

- box
- gaussian

Double1d[] filterParams [INPUT, OPTIONAL, default=no default value]

provides the numeric parameters for that characterize the filter, such as width and position.

This input parameter is an array that contains two Double1d arrays. Alternatively, the user can specify any of xFilterParams and/or yFilterParams

- The first element of this input array i.e. the first Double1d provides the parameters for the filter function along the x-axis,
- The second element of this input array i.e. the second Double1d provides the parameters for the filter function along the y-axis

For example, if the filter type is gaussian, the filterparam[0].get(0) contains the kernel length (in pixels) along the x-axis , and the kernel beam width in x-axis(in pixels).

Double1d[] filterParams [INPUT, OPTIONAL, default=no default value]

As a second example, if the filter type is box then param[0] contains contains just a Double1d with a single-element, the kernel length (in pixels).

Double1d xFilterParams [INPUT, OPTIONAL, default=no default value]

provides the numeric parameters that characterize the filter, such as its width and position, along the first axis.

Please note that the user can provide by means of the filterParams input a single array of two Double1d objects in case the user want to specify both xFilterParams and yFilterParams

Double1d yFilterParams [INPUT, OPTIONAL, default=no default value]

provides the numeric parameters for that characterize the filter, such as its width and position, along the second axis.

Please note that the user can provide by means of the filterParams input a single array of two Double1d objects in case the user want to specify both xFilterParams and yFilterParams

String1d datasetTypes [INPUT, OPTIONAL, default=String1d(["science"])]

name of the types of HifiSpectrumDatasets that contain the spectra observed by the instrument when aimed to the source.

The spectra used for the gridding are only those from the HifiSpectrumDataset's whose type is equal to the given dataset type and flagged as 'isLine' datasets.

String datasetType [INPUT, OPTIONAL, default="science"]

name of the type of HifiSpectrumDatasets that contain the spectra observed by the instrument when aimed to the source.

The spectra used for the gridding are only those from the HifiSpectrumDataset's whose type is equal to the given dataset type and flagged as 'isLine' datasets.

Int1d datasetIndices [INPUT, OPTIONAL, default=no default value]

indices of the HifiSpectrumDatasets to be used to retrieve spectra to perform the gridding into a cube.

This parameter can be used to grab any datasets within the HifiTimelineProduct.

If this is parameter is given the datasetType parameter will be ignored.

Boolean ignoreOffs [INPUT, OPTIONAL, default=true]

tells whether ignore datasets whose type matches the given datasetType(s) but which are not took from the source (isLine column of the HTP' summary table is false in those datasets) or, on the contrary, grab also them.

Integer apid [INPUT, OPTIONAL, default=no default value]

no default value,

Integer apid [INPUT, OPTIONAL, default=no default value]

provides the apid of a HifiTimelineProduct to be taken from the ObservationContext to create a cube or a number of cubes if that HifiTimelineProduct has a number of subbands.

PointingProduct pointing [INPUT, OPTIONAL, default=no default value]

used to derive the positions of the spectra, ignoring any PointingProduct given in the ObservationContext (observation parameter), if an ObservationContext input is also been given.

Please note that this parameter becomes mandatory if the HifiTimelineProduct lacks columns regarding spectra coordinates ("latitude", "longitude" coordinates) neither an ObservationContext or an OffsetsTable are provided to the task in order to compute the spectra positions.

SiamProduct siam [INPUT, OPTIONAL, default=no default value]

product containing alignment matrix of each detector, used to derive the positions of the spectra together with the PointingProduct.

This input parameter overrides any SiamProduct inside the any given ObservationContext input, if also been given.

Please note that this parameter becomes mandatory if the HifiTimelineProduct lacks columns regarding spectra coordinates ("latitude", "longitude" coordinates) and neither an ObservationContext or an OffsetsTable are provided to the task in order to compute the spectra positions.

TableDataset offsetsTable [INPUT, OPTIONAL, default=no default value]

auxiliary table which can be used to provide spectra coordinates.

this input is an alternative way to provide the spectra positions when neither the auxiliary products are available nor the latitude and longitude columns are already computed in the HifiSpectrumDatasets.

Cube cube [OUTPUT, OPTIONAL, default=no default value]

No default value.

First cube produced by this task, if several segments where processed, or the only cube produced if only one segment was specified

Cube[] cubes [OUTPUT, OPTIONAL, default=no default value]

No default value.

Array of cubes produced by this task, one per (specified) subband of the HifiTimelineProduct.

Double1d xPoints [OUTPUT, OPTIONAL, default=no default value]

No default value.

x-coordinates of points that define the regular grid.

Double1d yPoints [OUTPUT, OPTIONAL, default=no default value]

No default value.

Double1d yPoints [OUTPUT, OPTIONAL, default=no default value]
y-coordinates of points that define the regular grid

History

- 02-Jun-2009 First version. Based on DoCube. Is able to handle several subbands in parallel.
- 03-Jun-2009 Reviewed documentation of the parameters, added some examples.
- 12-Nov-2009 Reviewed documentation of the xFilterParams, yFilterParams added some examples.
- 12-Nov-2009 Reviewed documentation of the refPixelCoordinates.

1.42. DoHkCheck

Full Name:	herschel.hifi.pipeline.level0.DoHkCheck
Type:	Java Task -
Import:	from herschel.hifi.pipeline.level0 import DoHkCheck

Description

It checks the House Keeping contained in a HifiCalibrationDataset

(like the LoTrendTable, FptTrendTable) if the values are inside the thresholds contained in the CalHk-Thresh The result of that check are inserted as Columns "hk flag" in the HifiTimelineProuct passed.

Moreover for mixer bias parameters the mean and the variance of that parameter are inserted in the HifiTimelineProuct.

If some parameter is Out of limit inside a SpectrumDatset a BooleanParameter=false with the same name will be added to the MetaData.

Not all Dataset in the HifiTimelineProuct are checked, the default are: {"stab", "hc" , "hot", "cold", "science", "other"}; but they can be changed with using the input parameter "type"

Several utility method have been added to the Task to have more user friendly check of the Trend Table

For each HK a different bit in the "hk flag" is raised.

Example

Example 1: From jide/hype:

```
from herschel.hifi.pipeline.product import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
hk=AccessPacketTask()(obsid=obsid,apid=1026)
htp=HifiTimelineProduct(hdf,hk)
DoWbsScanCount(htp) #add the integration time in the HK
thr=CalHkThresh()# default
thrsetDefaultFpu();
fpu=FpuTrendTable(thr.mnemonics,hk) #create
doCheck(htp=htp,cal=fpu, thresholdH=thr)
flags=htp.get(1, -"science")["hk flag"].data #Is a Long1d
raisedFlag= doCheck.getFlag("HF_AH1_MXBIAS_C",flags) #utility method
```

API Summary

Jython Syntax
`See example below.`

Properties
`HifiTimelineProduct htp [IO, MANDATORY, default=No default value]`
`Cal cal [INPUT, HifiCalibrationDataset, default=MANDATORY]`
`Cal cal [INPUT, HkThresholds, default=MANDATORY]`

Limitations

No limitation.

API details

Properties

HifiTimelineProduct htp [IO, MANDATORY, default=No default value]
The HifiTimelineProduct which will be updated with the hk flag

Cal cal [INPUT, HifiCalibrationDataset, default=MANDATORY]
The dataset which contains the Trend Analysis of the parameters to be checked

Cal cal [INPUT, HkThresholds, default=MANDATORY]
The Product which contains the thresholds of the parameters to be checked

See also

- reference.

1.43. DoHrsBadChans

Full Name:	herschel.hifi.pipeline.hrs.DoHrsBadChans
Alias:	DoHrsBadChans
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsBadChans
Category:	HIFI pipeline task

Description

Step2a: Task which corrects the correlation functions of HRS if bad channels have been

detected.

Needs the CalHrsBadChans Product as input, and applies a correction on the correlation functions.

IMPORTANT NOTE: This task has not been yet implemented, and correction scenario are still under discussions.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsBadChans()(htp=htp, cal=CalHrsBadChans)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsBadChans(htp=htp, cal=calHrsBadChans)
</code>
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]

CalHrsBadChans **cal** [INPUT, MANDATORY, default=No default value]

Limitations

No limitation.

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]

The HifiTimelineProduct which contains the HrsSpectrumDatasets.

CalHrsBadChans cal [INPUT, MANDATORY, default=No default value]

The Product which contains the bad channels table.

See also

- reference.

History

- 2005-04-29 - OCJ: creation
- 2006-05-17 - OCJ: CompositeDataset replaced by HifiVectorDataset
- 2006-09-11 - OCJ: HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: HifiPipelineProduct replaced by HifiTimelineProduct

1.44. DoHrsCorrSP

Full Name:	herschel.hifi.pipeline.hrs.DoHrsCorrSP
Alias:	DoHrsCorrSP
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsCorrSP
Category:	HIFI pipeline task

Description

Step10: Task which corrects HRS spectra from IF non-linearity errors.

Gets the spectra and applies the following correcting factors:

- divide each value by #channels, except for 0 : divide by (#channels * 2)
- get the power value vSigma
- multiply each spectrum by Math.pow(HrsDefaultValues.THRESHOLDS_VALUE / vSigma, 2)

An additional parameter is provided to convert the spectrum in dB relatively to the mean spectrum.

- in_db = 1: convert the spectrum in dB
- in_db = 0: no conversion

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=CalHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=CalHrsPowCorr)
htp = DoHrsWindow()(htp=htp, window="hanning")
htp = DoHrsSymm()(htp=htp, zeros="multiple")
htp = DoHrsFFT()(htp=htp, algo="DFT")
htp = DoHrsSmooth()(htp=htp, smooth="none")
htp = DoHrsFreq()(htp=htp, model="column")
htp = DoHrsCorrSP()(htp=htp, in_db=0)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
doHrsWindow(htp=htp, window="hanning")
doHrsSymm(htp=htp, zeros="multiple")
doHrsFFT(htp=htp, algo="DFT")
doHrsSmooth(htp=htp, smooth="none")
```

Example 2: From jide (new style):

```
doHrsFreq(htp=htp, model="column")
doHrsCorrSP(htp=htp, in_db=0)
</code>
```

API Summary

Jython Syntax
See example below.

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
Integer **in_db** [INPUT, OPTIONAL, default=0]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]

The HifiTimelineProduct which contains the HrsSpectrumDatasets.

Integer in_db [INPUT, OPTIONAL, default=0]

The indicator to select if spectrum is converted in dB (in_db=1) or not (in_db=0).

History

- 2005-04-29 - OCJ: creation
- 2005-11-15 - OCJ: conversion in dB suppressed, see DoHrsDbSP
- 2006-05-17 - OCJ: CompositeDataset replaced by HifiVectorDataset
- 2006-09-11 - OCJ: HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: HifiPipelineProduct replaced by HifiTimelineProduct

1.45. DoHrsCutBandEdges

Full Name:	herschel.hifi.pipeline.hrs.DoHrsCutBandEdges
Alias:	DoHrsCutBandEdges
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsCutBandEdges
Category:	HIFI pipeline task

Description

Task which cuts the edges of the HRS sub-bands, according to the bandpass of the filter.

Gets as input the "cuts" frequency limits processed by DoHrsFreq, and truncates the sub-band edges, updates accordingly the MetaData subbandstart_xx and subbandlength_xx.

IMPORTANT NOTE: This Task creates the frequency columns even if a frequency model is used, and the model is automatically removed at the end of this Task.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=CalHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=CalHrsPowCorr)
htp = DoHrsWindow()(htp=htp, window="hanning")
htp = DoHrsSymm()(htp=htp, zeros="multiple")
htp = DoHrsFFT()(htp=htp, algo="DFT")
htp = DoHrsSmooth()(htp=htp, smooth="none")
htp = DoHrsFreq()(htp=htp, model="column")
htp = DoHrsCorrSP()(htp=htp, in_db=0)
htp = DoHrsCutBandEdges()(htp=htp)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
doHrsWindow(htp=htp, window="hanning")
doHrsSymm(htp=htp, zeros="multiple")
doHrsFFT(htp=htp, algo="DFT")
doHrsSmooth(htp=htp, smooth="none")
doHrsFreq(htp=htp, model="column")
doHrsCorrSP(htp=htp, in_db=0)
doHrsCutBandEdges(htp=htp)
</code>
```

API Summary

Jython Syntax
See example below.

Property
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]

API details

Property

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the HrsSpectrumDatasets.

History

- 2007-03-05 - OCJ: creation

1.46. DoHrsFFT

Full Name:	herschel.hifi.pipeline.hrs.DoHrsFFT
Alias:	DoHrsFFT
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsFFT
Category:	HIFI pipeline task

Description

Step9: Task which applies a FFT processing on the correlation functions of HRS.

Gets the type of FFT to apply, and applies it to the correlation functions to obtain HRS spectra. The type of FFT can be selected with:

- type = "DFT": Direct Fourier Transform
- type = "FFT": Fast Fourier Transform

IMPORTANT NOTE: no further calibrations can be made on correlation functions after this step, as they are replaced by spectra within the same columns.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=CalHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=CalHrsPowCorr)
htp = DoHrsWindow()(htp=htp, window="hanning")
htp = DoHrsSymm()(htp=htp, zeros="multiple")
htp = DoHrsFFT()(htp=htp, algo="DFT")
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
doHrsWindow(htp=htp, window="hanning")
doHrsSymm(htp=htp, zeros="multiple")
doHrsFFT(htp=htp, algo="DFT")
</code>
```

API Summary

Jython Syntax

```
See example below.
```

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
String **algo** [INPUT, OPTIONAL, default="DFT"]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the HrsSpectrumDatasets.

String algo [INPUT, OPTIONAL, default="DFT"]
A modifier for defining the type of Fourier Transform: "DFT" or "FFT"

History

- 2006-05-17 - OCJ: CompositeDataset replaced by HifiVectorDataset
- 2006-09-11 - OCJ: HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: HifiPipelineProduct replaced by HifiTimelineProduct

1.47. DoHrsFreq

Full Name:	herschel.hifi.pipeline.hrs.DoHrsFreq
Alias:	DoHrsFreq
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsFreq
Category:	HIFI pipeline task

Description

Step10: Task which computes the frequency of the sub-bands.

The input parameter "model" of this Task permits to choose between:

- "model" ==> creation of the HRS frequency Linear Model.
- "column" ==> creation of the frequency columns.

If the model is Chosen, the parameters of the model are the LOs values and the sampler name for every sub-band. Those parameters can change from scan to scan.

If the columns are chosen, this task will create columns named "frequency_1", "frequency_2", etc.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=CalHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=CalHrsPowCorr)
htp = DoHrsWindow()(htp=htp, window="hanning")
htp = DoHrsSymm()(htp=htp, zeros="multiple")
htp = DoHrsFFT()(htp=htp, algo="DFT")
htp = DoHrsSmooth()(htp=htp, smooth="none")
htp = DoHrsFreq()(htp=htp, model="column")
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
doHrsWindow(htp=htp, window="hanning")
doHrsSymm(htp=htp, zeros="multiple")
doHrsFFT(htp=htp, algo="DFT")
doHrsSmooth(htp=htp, smooth="none")
doHrsFreq(htp=htp, model="column")
</code>
```

API Summary

Jython Syntax
`See example.`

Properties
`HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]`
`String model [INPUT, OPTIONAL, default="column"]`

API details

Properties

`HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]`

The HifiTimelineProduct which contains the HrsSpectrumDatasets.

`String model [INPUT, OPTIONAL, default="column"]`

The selector between columns or frequency model ("column", "model").

History

- `2006-03-13 - OCJ`: creation
- `2006-05-17 - OCJ`: CompositeDataset replaced by HifiVectorDataset
- `2006-09-11 - OCJ`: HifiVectorDataset replaced by HifiPipelineProduct
- `2006-12-07 - OCJ`: HifiPipelineProduct replaced by HifiTimelineProduct

1.48. DoHrsNorm

Full Name:	herschel.hifi.pipeline.hrs.DoHrsNorm
Alias:	DoHrsNorm
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsNorm
Category:	HIFI pipeline task

Description

Step4: Task which normalizes the raw correlation functions of HRS.

Corrects the skew introduced by the internal multiplication table of HRS, and divides all channels by the first channel (channel0 of the rawCF).

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
</code>
```

API Summary

Jython Syntax
`See example below.`

Property
`HifiTimelineProduct` **`htp`** `[INOUT, MANDATORY, default=No default value]`

API details

Property

`HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]`

The HifiTimelineProduct which contains the HrsSpectrumDatasets.

History

- 2005-04-28 - OCJ: jide example corrected
- 2006-05-17 - OCJ: CompositeDataset replaced by HifiVectorDataset
- 2006-09-11 - OCJ: HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: HifiPipelineProduct replaced by HifiTimelineProduct

1.49. DoHrsOffsetPow

Full Name:	herschel.hifi.pipeline.hrs.DoHrsOffsetPow
Alias:	DoHrsOffsetPow
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsOffsetPow
Category:	HIFI pipeline task

Description

Step3: Task which computes the offset and power of HRS.

Gets science data, duration and offset from the HRS readout, and computes the offset (mSigma) and power (vSigma) of the analog input signal seen by the digital part of HRS.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
</code>
```

API Summary

Jython Syntax
See example below.

Property
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]

API details

Property

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]

The HifiTimelineProduct which contains the HrsSpectrumDatasets.

History

- `2005-04-27 - OCJ`: Column offset removed at the end of this step
- `2006-05-17 - OCJ`: CompositeDataset replaced by HifiVectorDataset
- `2006-09-11 - OCJ`: HifiVectorDataset replaced by HifiPipelineProduct
- `2006-12-07 - OCJ`: HifiPipelineProduct replaced by HifiTimelineProduct

1.50. DoHrsPowCorr

Full Name:	herschel.hifi.pipeline.hrs.DoHrsPowCorr
Alias:	DoHrsPowCorr
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsPowCorr
Category:	HIFI pipeline task

Description

Step6: Task which corrects the non-linearity error of the power of HRS.

Needs the CalHrsPowCorr calibration Product as input, and then corrects the input signal power with this Product.

IMPORTANT NOTE : this Task is mandatory. If the CalHrsPowCorr product is None or not valid, the spectra obtained at the end of the HRS pipeline are not valid.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=calHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=calHrsPowCorr)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
</code>
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]

CalHrsPowCorr **cal** [INPUT, MANDATORY, default=No default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the HrsSpectrumDatasets.

CalHrsPowCorr cal [INPUT, MANDATORY, default=No default value]
The Product which contains the power non-linearity correction table.

History

- `2005-04-28 - OCJ`: jide example corrected
- `2006-05-17 - OCJ`: CompositeDataset replaced by HifiVectorDataset
- `2006-09-11 - OCJ`: HifiVectorDataset replaced by HifiPipelineProduct
- `2006-12-07 - OCJ`: HifiPipelineProduct replaced by HifiTimelineProduct

1.51. DoHrsQDCFast

Full Name:	herschel.hifi.pipeline.hrs.DoHrsQDCFast
Alias:	DoHrsQDCFast
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsQDCFast
Category:	HIFI pipeline task

Description

Step5a: Task which corrects roughly the quantization distortion of the correlation functions.

Needs the CalHrsQDCFull calibration product as input, and multiply the channels by a correction factor, except for the channel0.

IMPORTANT NOTE : this Task can be used in place of DoHrsQDCFull only for performance reason, as for QLA for instance, but the quality of the spectra obtained at the end of the HRS pipeline is degraded.

Examples

Example 1: From jide: from herschel.hifi.pipeline.product import * from herschel.hifi.pipeline.hrs

```
import * hrs_frame = MkhrsDataset()(df_list=df, hk_list=meta) hrs_rawCF =
DoHrsSubbands()(htp=hrs_frame) hrs_RawCF_OffsetPow = DoHrsOffsetPow()
(htp=hrs_rawCF) hrs_normCF =
DoHrsNorm()(htp=hrs_RawCF_OffsetPow) calQDC = CalHrsQDCFast() hrs_corrCF =
DoHrsQDCFast()(htp=hrs_normCF, cal=calQDC)
```

Example 2: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFast()(htp=htp, cal=calHrsQDCFast)
</code>
```

Example 3: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFast(htp=htp, cal=calHrsQDCFast)
</code>
```

API Summary

Jython Syntax

```
See example below.
```

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
CalHrsQDCFast **cal** [INPUT, MANDATORY, default=No default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the HrsSpectrumDatasets.

CalHrsQDCFast cal [INPUT, MANDATORY, default=No default value]
The Product which contains the Fast Quantization Distortion Correction factor.

History

- 2005-04-28 - OCJ: : jide example corrected
- 2006-05-17 - OCJ: : CompositeDataset replaced by HifiVectorDataset
- 2006-09-11 - OCJ: : HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: : HifiPipelineProduct replaced by HifiTimelineProduct

1.52. DoHrsQDCFull

Full Name:	herschel.hifi.pipeline.hrs.DoHrsQDCFull
Alias:	DoHrsQDCFull
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsQDCFull
Category:	HIFI pipeline task

Description

Step5b: Task which corrects the quantization distortion of the correlation functions.

Needs the CalHrsQDCFull calibration product as input, and then interpolates on the 3 dimensional table of this product the correlation function for each correlation channel.

IMPORTANT NOTE 1: this Task is mandatory. If the CalHrsQDCFull product is None or not valid, the spectra obtained at the end of the HRS pipeline are not valid.

IMPORTANT NOTE 2: this Task can be replaced by DoHrsQDCFast only for performance reason, for QLA for instance, but the quality of the spectra obtained at the end of the HRS pipeline is degraded.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=calHrsQDCFull)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
</code>
```

API Summary

Jython Syntax
See example below.

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
CalHrsQDCFull **cal** [INPUT, MANDATORY, default=No default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the HrsSpectrumDatasets.

CalHrsQDCFull cal [INPUT, MANDATORY, default=No default value]
The Product which contains the Full Quantization Distortion Correction table.

History

- `2005-04-28 - OCJ`: jide example corrected
- `2006-05-17 - OCJ`: CompositeDataset replaced by HifiVectorDataset
- `2006-09-11 - OCJ`: HifiVectorDataset replaced by HifiPipelineProduct
- `2006-12-07 - OCJ`: HifiPipelineProduct replaced by HifiTimelineProduct

1.53. DoHrsSmooth

Full Name:	herschel.hifi.pipeline.hrs.DoHrsSmooth
Alias:	DoHrsSmooth
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsSmooth
Category:	HIFI pipeline task

Description

Step12: Task which applies a Hanning smoothing on the spectra, which is equivalent to a Hanning windowing on correlation functions.

An additional parameter is provided to define the type of smoothing to apply:

- "hanning": 1/4 (x-1) + 1/2 (x) + 1/4 (x+1)
- "none": keep HifiTimelineProduct without any change

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=CalHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=CalHrsPowCorr)
htp = DoHrsWindow()(htp=htp, window="hanning")
htp = DoHrsSymm()(htp=htp, zeros="multiple")
htp = DoHrsFFT()(htp=htp, algo="DFT")
htp = DoHrsSmooth()(htp=htp, smooth="none")
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
doHrsWindow(htp=htp, window="hanning")
doHrsSymm(htp=htp, zeros="multiple")
doHrsFFT(htp=htp, algo="DFT")
doHrsSmooth(htp=htp, smooth="none")
</code>
```

API Summary

Jython Syntax
`See example below.`

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
String **smooth** [INPUT, OPTIONAL, default="hanning"]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the HrsSpectrumDatasets.

String smooth [INPUT, OPTIONAL, default="hanning"]
A modifier for defining the type of smoothing to apply: "hanning", "none"

History

- 2007-06-21 - OCJ: : Creation

1.54. DoHrsSubbands

Full Name:	herschel.hifi.pipeline.hrs.DoHrsSubbands
Alias:	DoHrsSubbands
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsSubbands
Category:	HIFI pipeline task

Description

Step2: Task which extracts the HRS sub-bands from the HRS readout.

Gets the frames, the configurations of the HRS ASICs, and creates as many columns of type correlation functions as sub-bands found.

HRS provides 4080 channels, which can be affected to different samplers by changing the HRS configurations. This permits to have for science observation modes from 1 sub-band of 4080 channels to 8 sub-bands of 510 channels. Only Engineering modes can manage sub-bands of different sizes.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
</code>
```

API Summary

Jython Syntax

```
See example below.
```

Property

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]

API details

Property

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]

The HifiTimelineProduct which contains the HRS SpectrumDatasets.

History

- `2005-04-28 - OCJ`: : configuration removed from MetaData
- `2005-07-26 - OCJ`: : new columns for additional parameters added
- `2006-05-17 - OCJ`: : CompositeDataset replaced by HifiVectorDataset
- `2006-09-11 - OCJ`: : HifiVectorDataset replaced by HifiPipelineProduct
- `2006-12-07 - OCJ`: : HifiPipelineProduct replaced by HifiTimelineProduct

1.55. DoHrsSymm

Full Name:	herschel.hifi.pipeline.hrs.DoHrsSymm
Alias:	DoHrsSymm
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsSymm
Category:	HIFI pipeline task

Description

Step8: Task which symmetries the correlation function of HRS to prepare the FFT.

Gets the correlation functions, duplicates the channels except channel0, and inverse them.

A modifier allows the user to define the zero filling:

- "multiple" = multiple of 2 (DFT)
- "power" = power of 2 (FFT)

For a wide-band resolution sub-band, max number of zeroes can be 5 (1024 - (2*510 - 1)), for a high-resolution sub-band, max number of zeros can be 17.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=CalHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=CalHrsPowCorr)
htp = DoHrsWindow()(htp=htp, window="hanning")
htp = DoHrsSymm()(htp=htp, zeros="multiple")
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
doHrsWindow(htp=htp, window="hanning")
doHrsSymm(htp=htp, zeros="multiple")
</code>
```

API Summary

Jython Syntax
`See example below.`

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
String **zeros** [INPUT, OPTIONAL, default="multiple"]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the HrsSpectrumDatasets.

String zeros [INPUT, OPTIONAL, default="multiple"]
A modifier for defining the zero filling.

History

- 2005-04-28 - OCJ: jide example corrected
- 2006-05-17 - OCJ: CompositeDataset replaced by HifiVectorDataset
- 2006-09-11 - OCJ: HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: HifiPipelineProduct replaced by HifiTimelineProduct

1.56. DoHrsWindow

Full Name:	herschel.hifi.pipeline.hrs.DoHrsWindow
Alias:	DoHrsWindow
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import DoHrsWindow
Category:	HIFI pipeline task

Description

Step7: Task which applies a Hanning or Hamming windowing to the correlation functions of HRS, which is equivalent to a Hanning smoothing on spectra.

An additional parameter is provided to define the type of smoothing to apply:

- "hanning": Hanning window on correlation function
- "none": keep HifiTimelineProduct without any change

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
htp = DoHrsOffsetPow()(htp=htp)
htp = DoHrsNorm()(htp=htp)
htp = DoHrsQDCFull()(htp=htp, cal=CalHrsQDCFull)
htp = DoHrsPowCorr()(htp=htp, cal=CalHrsPowCorr)
htp = DoHrsWindow()(htp=htp, window="hanning")
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsOffsetPow(htp=htp)
doHrsNorm(htp=htp)
doHrsQDCFull(htp=htp, cal=calHrsQDCFull)
doHrsPowCorr(htp=htp, cal=calHrsPowCorr)
doHrsWindow(htp=htp, window="hanning")
</code>
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]

Properties
String **window** [INPUT, OPTIONAL, default="hanning"]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]

The HifiTimelineProduct which contains the HrsSpectrumDatasets.

String window [INPUT, OPTIONAL, default="hanning"]

A modifier for defining the type of window to apply.

History

- 2005-04-28 - OCJ: jide example corrected
- 2006-05-17 - OCJ: CompositeDataset replaced by HifiVectorDataset
- 2006-09-11 - OCJ: HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: HifiPipelineProduct replaced by HifiTimelineProduct

1.57. DoMainBeamTemp

Full Name:	herschel.hifi.pipeline.generic.DoMainBeamTemp
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoMainBeamTemp

Description

Provides the main beam temperature.

Example

Example 1: From jide:

```
doMainBeamTemp(htp=htp, cal=my_beam_eff)
doMainBeamTemp(htp=htp, beamEff=0.68)
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

GenericPipelineCalProduct **cal** [INPUT, OPTIONAL, default=no default value]

Double **beamEff** [INPUT, OPTIONAL, default=no default value]

MapContext **calibration** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

GenericPipelineCalProduct cal [INPUT, OPTIONAL, default=no default value]

A calibration product that provides the beam efficiencies. The product allows for LO-specific parameters.

Double beamEff [INPUT, OPTIONAL, default=no default value]

The beam efficiency to apply.

MapContext calibration [INPUT, OPTIONAL, default=no default value]

Parameter to pass a calibration context from which all the calibration input, here the beam efficiencies, can be retrieved.

History

- `2009-09-09 - meli`: : new implementation.

1.58. DoOffSubtract

Full Name:	herschel.hifi.pipeline.generic.DoOffSubtract
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoOffSubtract

Description

Computes the differences between ON and OFF positions - the way of how this subtraction is

processed depends on the observing mode.

- Either, the values found in a baseline calibration product are suitably interpolated in time (obsTime) and subtracted from the source measurements (FSwitch-, LoadChop-, MappingOTF-modes.)
- Or, the off position also contains signal but with reversed optical paths (and standing waves with opposite sign) so that the ON and the OFF datasets are basically averaged on a scan-by-scan basis (DBS- and FastDBS-modes).
- Or, the the off scans are subtracted on a scan-by-scan basis (PositionSwitch-mode)

A validation mechanism checks the spectra to be subtracted from each other and, if needed, sets in the result data a row flag. The default mechanism inspects the mixer currents (columns 'MJC_hor' or 'MJC_Ver', respectively) and tests their relative deviation (of the ON and the OFF spectra) to be less than a given tolerance. Once the tolerance is exceeded, for the result spectrum a row flag (bit 17) is set. The tolerance can be set by setting the 'validatorTolerance'-parameter. By default it is set to 0.025. This default checking mechanism can be overruled by passing as 'validator'-parameter an instance of the {@link DataValidator} interface. When setting this parameter to None, no checking is done.

A filter options allows to ignore flagged spectra found in the baselines. Two parameters are typically set here:

- Boolean parameter `ignoreFlagged`: if set to true flagged scans (rowflag) contained in the baseline are ignored.
- `bits` : Specify the bits the rowflag should be tested for. Once one of the given bits are set in the flag value the corresponding scan is ignored. Either you can specify an integer value or a (Py-)list of such.

Note that if the result of this filtering would be that no baseline would be left no filtering is done.

Examples

Example 1: From jide:

```
# the mode typically used for DBS modes
htp = doOffSubtract(htp=spectra, mode='addOff')
```

Example 2: From jide:

```
# baseline: a CalOffBaseline-product typically generated by the MkOffSmooth
step
htp = doOffSubtract(htp=spectra, mode='subtractBaseline', cal=baseline)
```

Example 3: From jide:

```
# baseline: a CalOffBaseline-product typically generated by the MkOffSmooth
step
```

Example 3: From jide:

```
htp = doOffSubtract(htp=spectra, mode='subtractBaseline',
interpolator='NEXT', cal=baseline)
```

Example 4: From jide:

```
# the validator tolerance is re-set.
htp = doOffSubtract(htp=spectra, cal=baseline, validatorTolerance=0.1)
```

Example 5: From jide:

```
# validation is switched off.
htp = doOffSubtract(htp=spectra, cal=baseline, validator=None)
```

Example 6: From jide:

```
# filter out scans in the OFF datasets with non-zero row flags.
htp = doOffSubtract(htp=spectra, cal=baseline, ignoreFlagged=True)
```

Example 7: From jide:

```
# filter out scans in the OFF datasets with row flags that at least one of
the specified bits set.
htp = doOffSubtract(htp=spectra, cal=baseline, ignoreFlagged=True, bits=[8,15])
```

Example 8: From jide:

```
# an alternative validation mechanism is set (here based, as dummy, on the
LoFrequency)
from herschel.hifi.pipeline.generic.dorefsubtract import
SubtractionDataValidatorImpl
validator = SubtractionDataValidatorImpl()
validator.configure('LoFrequency', 0.001, True)
htp = doOffSubtract(htp=spectra, cal=baseline, validator=validator)
```

API Summary

Properties
Object **validatorTolerance** [INPUT, OPTIONAL, default=0.025]
DataValidator **validator** [INPUT, OPTIONAL, default=no default value]
MapContext **calibration** [INPUT, OPTIONAL, default=no default value]
Boolean **ignoreFlagged** [INPUT, OPTIONAL, default=False]
Object **bits** [INPUT, OPTIONAL, default=no default value]

API details

Properties

Object validatorTolerance [INPUT, OPTIONAL, default=0.025]

The tolerance to be used for the default validator that checks the difference in the mixer currents and sets a row flag once the mixer currents of the ON and OFF deviate by more than the tolerance from each other. A relative distance measure is used to quantify the deviation (`2abs(x1-x2)/(abs(x1)+abs(x2))`). Alternatively, a calibration object of type

Object validatorTolerance [INPUT, OPTIONAL, default=0.025]

`GenericPipelineCalProduct`can be passed from which the band-specific tolerances will be extracted. Yet another possibility is to pass the whole calibration context - the task retrieves the appropriate product with the tolerances. Furthermore, you can use the parameter 'calibration' for that purpose. Once a product of type `GenericPipelineCalProduct` is obtained, the appropriate tolerance is looked up from a table (associated with the given date) from a column named 'doOffSubtract'.

DataValidator validator [INPUT, OPTIONAL, default=no default value]

Custom validator that can be passed to the task to check source and ref that is subtracted from each other and flag data once the tolerance exceeds a given tolerance. Note that the validatorTolerance cannot be used to specify this threshold. Furthermore, the default validator is overwritten once a validator or None is passed here.

MapContext calibration [INPUT, OPTIONAL, default=no default value]

Parameter to pass a calibration context from which all the calibration input, here just mixer current tolerances, can be retrieved.

Boolean ignoreFlagged [INPUT, OPTIONAL, default=False]

Ignore specific scans in the baseline data that have one of the bits set in the row flag as specified with the `bits`-parameter. If the `bits` is not set all the non-zero row flags are ignored.

Object bits [INPUT, OPTIONAL, default=no default value]

The bits to be tested for in the row flags so that once one of those is present in a given row flag value the corresponding row is ignored (once the parameter ignoreFlagged is set to True. Note that this option only applies in combination with the `mode=subtractBaseline`. The bits are specified as an integer or a list of integers.

History

- 2007-06-28 meli: New version.
- 2007-11-29 meli: Javadoc updated.
- 2008-04-02 meli: Handle different LoFrequency groups.
- 2009-08-10 meli: Add interpolation parameter.
- `2009-09-23 - meli`: Validation mechanism.
- `2009-11-24 - meli`: Filter for row flags with specific bits set.

1.59. DoRadialVelocity

Full Name:	herschel.hifi.pipeline.generic.DoRadialVelocity
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoRadialVelocity

Description

Corrects the frequency 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 are "HSO" (short for Herschel Space Observatory), "LSR" or "SOURCE" (for the rest frame of the source). Adopting a non-relativistic approach only the radial velocity is needed, i.e. only the component in direction of the pointing of the telescope. This can be obtained from different sources:

- from an object of type herschel.share.fltdyn.ephem.Ephemerides passed as 'ephem' parameter to the task;
- from an auxiliary context passed as 'aux' parameter to the task;
- from the datasets to be processed, in a column named 'velocity';
- from the task parameter velocity or velocity_source - only to specify the velocity of the source relative to LSR, constant in time.

By default, a positive value entered as velocity_source parameter means that the source and S/C depart from each other (redshift). Note that a positive velocity found in the 'velocity' column (which is the velocity of the S/C w.r.t. LSR) has just the opposite effect. 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. For this to happen you can set the parameter reverse=True. You cannot repeatedly correct for different source velocities. In order to try another source velocity do the reverse correction first and appply the modified correction thereafter. The frame to transform to is by default LSR for non-SSO's and SOURCE for SSO's. in case you want to depart from this default (e.g. transform to the rest frame of the source for non-SSO's) you can specify that by setting the targetFrame' parameter. Note that in contrast to earlier versions (<=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.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoRadialVelocity
doRadialVelocity = DoRadialVelocity()
# apply the defauls -- transform to the LSRk frame, assume that the radial
velocity is contained in the spectra.
doRadialVelocity(htp=htp)
# transform back to the rest frame of the s/c, assume that the radial
velocity of the s/c is contained in the spectra.
doRadialVelocity(htp=htp, targetFrame = -"source", velocity_source=10.0) # in
km/sec
# transform back to the rest frame of the s/c
doRadialVelocity(htp=htp, reverse=True)
# transform to the rest frame of the source
doRadialVelocity(htp=htp, targetFrame = -"source", aux=obs.auxiliary,
velocity_source=10.0)
# transform to the rest frame of the source
```

Example 1: From jide:

```
doRadialVelocity(htp=htp, targetFrame = -"source", aux=obs.auxiliary,
velocity_source=10.0)
```

API Summary

Jython Syntax

See example below.

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

String **targetFrame** [IN, MANDATORY, default=no default value]

Double **velocity_source** [IN, OPTIONAL, default=no default value]

Double **velocity** [IN, OPTIONAL, default=no default value]

Object **ephem** [IN, OPTIONAL, default=no default value]

MapContext **aux** [IN, OPTIONAL, default=no default value]

Boolean **reverse** [IN, OPTIONAL, default=False]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

String targetFrame [IN, MANDATORY, default=no default value]

The rest frame to transform to.

Double velocity_source [IN, OPTIONAL, default=no default value]

Velocity (in km per second) to be used in the velocity correction. If this parameter is specified both the velocity of the S/C and the specified velocity_source is corrected for. The specified velocity is included as meta data to the datasets and the timeline product.

Double velocity [IN, OPTIONAL, default=no default value]

The same as 'velocity_source'.

Object ephem [IN, OPTIONAL, default=no default value]

Pass an ephemeris object or and OrbitEphemeris-product from which the velocity information can be extracted. Once this parameter is set, the velocity information found in the ephemeris object is used to compute the radial velocity of the spacecraft wrt LSR (or the SSO rest frame) and the Doppler correction. The velocity computed and used here is added to the datasets (column name "velocity"). If velocity data has already been entered into the data this information is overwritten.

MapContext aux [IN, OPTIONAL, default=no default value]

An auxiliary context from which the OrbitEphemeris product can be extracted. This is used (once the 'ephem' parameter is not set) for computing the radial velocity of the spacecraft

MapContext aux [IN, OPTIONAL, default=no default value]

wrt LSR. For SSO's, the current OrbitEphemeris-product does not contain a all the planetary ephemeris data. If velocity data has already been entered into the data this information is over-written.

Boolean reverse [IN, OPTIONAL, default=False]

By setting this parameter to true you can revert all the Doppler-shift correction applied so far. The cumulative effect of all Doppler corrections can be obtained by comparing the tuned with the adjusted LO frequencies. In case only a single LO frequency column is found that task assumes that no correction has been applied so that there is nothing to revert. Since we assume that the original frame is 'HSO' (or topocentric) 'reverse=True' is equivalent to setting "targetFrame='HSO'".

1.60. DoRefSubtract

Full Name:	herschel.hifi.pipeline.generic.DoRefSubtract
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoRefSubtract

Description

Used for the reference subtraction in which measurements from a suitable reference position

(reference sky position, cold load or with a switched LO frequency) are subtracted from the source - the way how to identify source and reference measurement scans depends on the particular AOT / observing mode applied.

In all cases, only the science data is involved - but typically both, the ON and the OFF measurements.

- Position switch reference modes:

  Here the reference position is obtained by moving the telescope to a different blank sky position. In the pipeline processing, this is treated rather with the DoOffSmooth module. So the application of the DoRefSubtract here has no impact.

- Slow chop DBS modes:

  In ON and OFF datasets, we expect a pattern of the form ref-source-source-ref-... for ON and source-ref-ref-source-source-... for the OFF spectra. The subtraction is carried through pairwise for each subsequent source/ref-pair.

- Fast chop DBS modes:

  In ON and OFF datasets, we expect a pattern of the form ref-source-ref-source-ref-... for ON and source-ref-source-ref-source-... for the OFF spectra. The subtraction is carried through pairwise for each subsequent source/ref-pair.

- Load chop modes:

  The reference position is the cold load. Again, a pattern of the form ref-source-source-ref-ref-... is assumed that should be reflected by the Chopper value. Again the subtraction is carried through pairwise for each sub-sequent source/ref-pair.

- Frequency switch mode: The reference is not a different position but rather a different setting for the LO frequency. Here, a pattern of the form source-ref-ref-source-... is assumed (i.e. starting with 'source', the prime LO frequency) and the subtraction is carried through pairwise for each sub-sequent source/ref-pair.

Different configuration possibilities are available:

- Assume a pattern of the form A-B-B-A or A-B-A-B: Specify indicator="pattern" and set the isABBA-parameter to true or false, respectively. Furthermore, specify the startsWithRef and the offStartsWithOpposite-flags.

- Use the "buffer"-values and map its values to 'source' or 'ref': Specify indicator="buffer" and either

  - set the startsWithRef: in this case, the first value found (within a dataset) is associated with the reference or the source depending on whether it has been set to true or false.

  - specify a dictionary of the form {"source":2,"ref":1} to map the source and ref to specific buffer values.

- Use the "LoFrequency"- or the "Chopper"-values and map its values to 'source' or 'ref': Specify indicator="LoFrequency" or "Chopper", respectively and either
  - set the startWithRef: in this case, the first value found (within a dataset) is associated with the reference or the source depending on whether it has been set to true or false.
  - specify a dictionary of the form {"source":-4.1,"ref":4.8} to map the source and ref to specific values found in the "LoFrequency" or "Chopper"-column. Note that for these double values, internally set measurement errors are used for the LoFrequency and the ChopperPositions.

A validation mechanism checks the spectra to be subtracted from each other and, if needed, sets in the result data a row flag. The default mechanism inspects the mixer currents (columns 'MJC_hor' or 'MJC_Ver', respectively) and tests their relative deviation (of the source and ref spectra) to be less than a given tolerance. Once the tolerance is exceeded, for the result spectrum a row flag (bit 16) is set. The tolerance can be set by setting the 'validatorTolerance'-parameter. By default it is set to 0.025. This default checking mechanism can be overruled by passing as 'validator'-parameter an instance of the {@link DataValidator} interface. When setting this parameter to None, no checking is done.

Examples

Example 1: From jide:

```
htp = doRefSubtract(htp=spectra, isABBA=True, startsWithRef=True,
offStartsWithOpposite=True)
```

Example 2: From jide:

```
htp = doRefSubtract(htp=spectra, indicator ="buffer",
phaseValues={"source":2,"ref":1}, offStartsWithOpposite=True)
```

Example 3: From jide:

```
from herschel.hifi.pipeline.generic.utils import ChopperPosition
htp = doRefSubtract(htp=spectra, indicator ="Chopper",
phaseValues={"source":ChopperPosition.CENTER,"ref":ChopperPosition.COLD},
offStartsWithOpposite=True)
```

Example 4: From jide:

```
# the validator tolerance is re-set.
htp = doRefSubtract(htp=spectra, isABBA=True, startsWithRef=True,
offStartsWithOpposite=True, validatorTolerance=0.1)
```

Example 5: From jide:

```
# validation is switched off.
htp = doRefSubtract(htp=spectra, isABBA=True, startsWithRef=True,
offStartsWithOpposite=True, validator=None)
```

Example 6: From jide:

```
# an alternative validation mechanism is set (here based, as dummy, on the
LoFrequency)
from herschel.hifi.pipeline.generic.dorefsubtract import
SubtractionDataValidatorImpl
validator = SubtractionDataValidatorImpl()
validator.configure('LoFrequency', 0.001, True)
htp = doRefSubtract(htp=spectra, isABBA=True, startsWithRef=True,
offStartsWithOpposite=True, validator=validator)
```

API Summary

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]
String **indicator** [INPUT, OPTIONAL, default=no default value]
Boolean **isABBA** [INPUT, OPTIONAL, default=no default value]
PyDictionary **phaseValues** [INPUT, OPTIONAL, default=no default value]
Boolean **startWithRef** [INPUT, OPTIONAL, default=no default value.]
Boolean **offStartsWithOpposite** [INPUT, OPTIONAL, default=no default value]
Object **validatorTolerance** [INPUT, OPTIONAL, default=0.025]
DataValidator **validator** [INPUT, OPTIONAL, default=no default value]
MapContext **calibration** [INPUT, OPTIONAL, default=no default value]
Boolean **ignore** [INPUT, OPTIONAL, default=no default value.]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The htp to be processed.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Configuration parameter that can be passed to the product

String indicator [INPUT, OPTIONAL, default=no default value]

The indicator from which source and ref can be identified. Either a column name ("Chopper", "LoFrequency", "buffer") or "pattern".

Boolean isABBA [INPUT, OPTIONAL, default=no default value]

Assume a A-B-B-A pattern if set to True - otherwise, a A-B-A-B pattern is taken.

PyDictionary phaseValues [INPUT, OPTIONAL, default=no default value]

Specify how the values found in the indicator column should be identified with "source" and "ref".

Boolean startWithRef [INPUT, OPTIONAL, default=no default value.]

Assumes that the sequence starts with a reference measurement if set to true (for DBS modes: ON datasets start with ref, OFF with signal).

Boolean offStartsWithOpposite [INPUT, OPTIONAL, default=no default value]

Flag to be used for DBS and FastDBS modes where the meaning of the source/ref change when going to from ON to OFF.

Object validatorTolerance [INPUT, OPTIONAL, default=0.025]

The tolerance to be used for the default validator that checks the difference in the mixer currents and sets a row flag once the mixer currents of the source and ref deviate by more than the tolerance from each other. A relative distance measure is used to quantify the deviation (`2abs(x1-x2)/(abs(x1)+abs(x2))`). Alternatively, a calibration object of type `GenericPipelineCalProduct` can be passed from which the band-specific tolerances will be extracted. Yet another possibility is to pass the whole calibration context - the task retrieves the appropriate product with the tolerances. Furthermore, you can use the parameter 'calibration' for that purpose. Once a product of type `GenericPipelineCalProduct` is obtained, the appropriate tolerance is looked up from a table (associated with the given date) from a column named 'doRefSubtract'.

DataValidator validator [INPUT, OPTIONAL, default=no default value]

Custom validator that can be passed to the task to check source and ref that is subtracted from each other and flag data once the tolerance exceeds a given tolerance. Note that the validatorTolerance cannot be used to specify this threshold. Furthermore, the default validator is overwritten once a validator or None is passed here.

MapContext calibration [INPUT, OPTIONAL, default=no default value]

Parameter to pass a calibration context from which all the calibration input, here mixer current tolerances and possibly chopper positions, can be retrieved.

Boolean ignore [INPUT, OPTIONAL, default=no default value.]

Ignore the execution of this module (as e.g. for position switch modes)

History

- `2007-05-16 - meli`: New version.
- `2007-11-14 - meli`: Javadoc updated.
- `2008-04-08 - meli`: Some minor changes.
- `2009-09-23 - meli`: Validation mechanism.

1.61. DoSidebandGain

Full Name:	herschel.hifi.pipeline.generic.DoSidebandGain
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoSidebandGain

Description

Apply the sideband gain coefficients to the flux values.

Using the 'sidebands' parameter you can specify to what sideband the data should be transformed to. The output products can be retrieved from the task as parameters 'htp' and/or 'image'. Note that the original timeline product 'htp' is modfied under this operation. The sideband gain coefficients are retrieved from the helper object of type {@link CalSidebandCoeff} that typically is composed by the {@link MkSidebandGain}-task. Note that this helper object does not explicitly on band or date information. Hence this information should (typically) be passed to the provider of the helper object (i.e. {@link MkSidebandGain}-task which accepts the {@link HifiTimelineProduct} as input). In case no such helper class is provided the default coefficients 0.5 are applied.

Example

Example 1: From jide: from herschel.hifi.pipeline.generic import DoSidebandGain

```
doSidebandGain = DoSidebandGain()
htpUpper = doSidebandGain(htp=spec, cal=gains)
htpLower = doSidebandGain.image
doSidebandGain(htp=spec, cal=gains, sideband='upper')
htpUpper = doSidebandGain.htpUpper
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [IO, MANDATORY, default=no default value]

HifiTimelineProduct **image** [OUT, OPTIONAL, default=no default value]

String **sideband** [IN, OPTIONAL, default='both' Determines to what]

CalSidebandCoeff **cal** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [IO, MANDATORY, default=no default value]

The timeline product to be processed and the timeline product processed with the coefficients of the selected sideband. In case both sidebands should be returned, we here return the upper sideband.

HifiTimelineProduct image [OUT, OPTIONAL, default=no default value]

The timeline product with flux processed according to the lower sideband in case both sidebands have been selected.

String sideband [IN, OPTIONAL, default='both' Determines to what]

sideband the flux and frequency values should be transformed to. Possible values are 'both', 'lower', 'upper'. Default value is 'both'.

CalSidebandCoeff cal [INPUT, OPTIONAL, default=no default value]

Provides the gains coefficients that should be applied to the flux values. The information on sideband, LO frequency and the IF frequency grid is provided so that IF frequency dependent gain coefficients can be obtained.

History

- `2005-09-23 - JCG`: Initial, change of name to DoSidebandGain (SPR-397).
- `2009-04-01 - Melchior`: Initial implementation.
- `2009-06-24 - Melchior`: Remove change in frequency scale.
- `2009-08-28 - Melchior`: New way of passing / retrieving the sideband gains coefficients.

1.62. DoStitch

Full Name:	herschel.hifi.pipeline.generic.DoStitch
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoStitch

Description

Task for stitching 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 iterates through the datasets found in the timeline product and delegates the processing to the {@link Stitch-Spectrum}-task in the spectrum toolbox. The stiched spectra originating from the same dataset are included in a new dataset. Since the shape of these spectra may be different after stitching the spectra need to be resampled. 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.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoSubBstitch
stitch = DoSubBstitch()
stitched11 = stitch(htp=htp)
# cut at crossover points and concatenate
stitched12 = stitch(htp=htp, variant="crossoverPoints")
# cut at crossover points and concatenate, consider crossover points only by
10 percent off the border to of the overlapping region
stitched13 = stitch(htp=htp, variant="crossoverPoints", edgeTolerance=0.1)
# cut at crossover points and concatenate, resample the resulting spectra to
1MHz channel width
stitched14 = stitch(htp=htp, variant="crossoverPoints", edgeTolerance=0.1,
stepsize=1.0, unit="MHz")
# cut at mid points of the overlapping regions and concatenate
stitched21 = stitch(htp=htp, variant="midPoints")
# cut at mid points of the overlapping regions and concatenate, resample to
1.0 channel width (in units of the underlying frequency scale)
stitched22 = stitch(htp=htp, variant="midPoints", stepsize=1.0)
# cut at predefined split points and concatenate
stitched31 = stitch(htp=htp, variant="splitPoints", splitPoints = [5000.0,
6000.0, 7000.0])
# cut at predefined split points and concatenate, resample to 1.0 channel
width (in units of the underlying frequency scale)
stitched32 = stitch(htp=htp, variant="splitPoints", splitPoints = [5000.0,
6000.0, 7000.0], stepsize=1.0)
# cut at predefined split points and concatenate, resample to 0.001 GHz
channel width
stitched33 = stitch(htp=htp, variant="splitPoints", splitPoints = [5.0, 6.0,
7.0], stepsize=0.001, unit="GHz")
# average the parts of the spectra that are overlapping, resample to a 1.0
channel width (in units of the underlying frequency scale)
stitched4 = stitch(htp=htp, variant="average", stepsize=1.0)
# average the parts of the spectra that are overlapping, resample to a 1.0
MHz channel width, do not consider flags and weights
# when computing teh average (but propagate them)
stitched41 = stitch(htp=htp, variant="average", stepsize=1.0, unit="MHz",
avg_variant="flux")
# average the parts of the spectra that are overlapping, avoid resampling as
far as possible
stitched42 = stitch(htp=htp, variant="average", avg_variant="flux")
```

Example 1: From jide:

```
# average the parts of the spectra that are overlapping, avoid resampling as
far as possible
stitched5 = stitch(htp=htp, selection_meta = ["science", -"hc"])
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value.]
String **variant** [INPUT, OPTIONAL, default="crossoverPoints".]
double **edgeTolerance** [INPUT, OPTIONAL, default=0.01.]
Object **splitPoints** [INPUT, OPTIONAL, default=no default value]
double **stepsize** [INPUT, OPTIONAL, default=no default value.]
String **unit** [INPUT, OPTIONAL, default=no default value.]
String **avg_variant** [INPUT, OPTIONAL, default=no default value.]
Object **selection_meta** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value.]

The input timeline product with the spectra to be stitched.

String variant [INPUT, OPTIONAL, default="crossoverPoints".]

The parameter that specifies the way how the stitching should be done. Possible values:

- "crossoverPoints": In overlapping ranges of the spectra a crossover point is taken to split the spectra. In case no crossover point is found a point is taken where the spectra come closest. In case many crossover points are found the one closest to the midpoint of the overlapping range is taken. The range where the crossover points are searched for is reduced by setting a non-zero edge tolerance (see parameter `edgeTolerance` below). This option only works if at most two segments 'participate' in the same overlapping range.
- "midPoint": In the overlapping ranges of the spectra the mid point is taken to split the spectra. Similar to the above, this option only works if at most two segments 'participate' in the same overlapping range.
- "splitPoints": In the overlapping ranges of the spectra previously specified points are taken to split the spectra (parameter `splitPoints`). the mid point is taken. Similar to the above, this option only works if at most two segments 'participate' in the same overlapping range.
- "average": In the overlapping ranges the average of the involved spectra is taken. This option allows to stitch spectra in which more than two spectra contribute to the same overlapping part.

double edgeTolerance [INPUT, OPTIONAL, default=0.01.]

Reduce the range (an overlapping range in the spectra) in which cross over points are searched. With `length` denoting the length of the original overlapping range `[n,m]` (in number of pixels) the reduced range is defined by `[n+p*length, m-p*length]` where `p` is the edge tolerance.

Object splitPoints [INPUT, OPTIONAL, default=no default value]

Specify the list of split points explicitly. This parameter is used only in combination with the option `variant="splitPoints"`. The split points are specified as a Python list. Note that the number of split points shoudl correspond to the number of overlapping ranges (per point spectrum).

double stepsize [INPUT, OPTIONAL, default=no default value.]

Specify the stepsize used to define a linear frequency the spectra are resampled to. The resampling is important in the following situations:

- When including the stitched point spectra in a common container. In case a `stepsize>0` is specified, a frequency grid is constructed with the given step size. The upper and lower bounds are determined from the minimum and maximum frequencies found in all the stitched point spectra (for a give segment index). In case a `stepsize=0` is specified, the frequency grid found in the first (stitched) point spectrum is taken as the output grid.
- When using `variant="average"` the spectra contributing to common ovrlapping frequency ranges are resampled to a common frequency grid.

When searching for crossover points, one of the spectra also needs to be resampled to a common frequency scale - here, the frequency scale of the other spectrum is used, though so that the `stepsize` parameter is not needed here.

String unit [INPUT, OPTIONAL, default=no default value.]

The unit the stepsize and/or split points are expressed in. In case no unit is specified the stepsize and/or the split points are assumed to be in the same units as the underlying frequency grid.

String avg_variant [INPUT, OPTIONAL, default=no default value.]

The variant of the average to be used: whether to include flags and weights in the processing.

Object selection_meta [INPUT, OPTIONAL, default=no default value]

Allows to specify a selection of datasets included in the timeline product by just looking at specific meta data values. Basically, there are two possibilities:

- As a py dictionary with keys specifying the meta data key and values the admissible values. These values are specified as lists.
- As a list of admissible 'sds_type's (the type appearing in the summary table). Hence this is internally translated into a py dictionary with 'sds_type' set as key. In order to distinguish ON and OFF science datasets, wee (artificially) allow for the values "scienceOn" and "scienceOff".

History

- 11-DEC-2009 Melchior : Initial version.

1.63. DoTimeCorr

Full Name:	herschel.hifi.pipeline.level0.DoTimeCorr
Type:	Java Task -
Import:	from herschel.hifi.pipeline.level0 import DoTimeCorr

Description

This task corrects the observation time (especially for HRS) because the

original obstime is the time of the last integration. As HRS can have several integrations where WBS generally only has 1, there exists a discrepancy between the obstime in HRS and WBS.

This task tries to correct for this effect, by correcting the obstimes to the beginning of the first integration. When a backend has only 1 integration, no correction is applied. This is true for most WBS measurements, and for quite some of HRS.

The formula in use is

obs_time = obs_time - f * (del + t_acc + 12ms)*(n_integr - 1)

where

n_integr is the number of integrations.

t_acc derives form "duration" for HRS and from "scancount" for WBS.

del is a delay 40ms for fastchop bbids (3225, 6042, 6043, 6053 and 6054) and 1ms otherwise. f is a parallel factor: 2 for fastchop and 1 otherwise.

When a correction is applied, the description field of the obstime is appended with " (corrected)".

The task is part of the Level0Pipeline.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoTimeCorr
doTimeCorr = DoTimeCorr()
doTimeCorr( htp=htp -)
```

API Summary

Jython Syntax

```
See example below.
```

Property

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

API details

Property

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]
The timeline product (observation) to be passed to the module.

History

- 2009-11-12 DK

1.64. DoUplink

Full Name:	herschel.hifi.pipeline.level0.DoUplink
Type:	Java Task -
Import:	from herschel.hifi.pipeline.level0 import DoUplink

Description

Copies several items from the UplinkProduct into the HTP and HSDS.

The following items can appear in the MetaData. It is also given from which Uplink items they might be copied.

- vlsr: the V_{lsr}, Velocity in the frame of reference.

  Taken from: redshift

- frame: the name of the reference frame.

  Taken from: frame

- raNominal: Requested Right Ascension.

  Taken from: ra

- decNominal: Requested Declination.

  Taken from: dec

- nyquistSampling: Map is Nyquist sampled.

  Taken from: flyNyquistSel|rasterNyquist|fsNyquist|LoadRasterNyquist

- pattAngle: Map rotation angle.

  Taken from: flyAngle|rasterAngle|fsAngle|LoadRasterAngle

- crossStep: Separation between scans of the map.

  Taken from: flyCrossStep|flyCrossScanStep|rasterCrossScanStep|fsCrossScanStep|crossStepSize

- mapWidth: Size of the map along the horizontal axis of the map.

  Taken from: flyX

- mapHeight: Size of the map along the vertical axis of the map.

  Taken from: flyY

- decoff: Declination of the reference point.

  Taken from: decoff

- raoff: Right ascension of the reference point.

  Taken from: raoff

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.level0 import DoUplink
```

Example 1: From jide:

```
doUplink = DoUplink()
doUplink( htp=htp, auxiliary=aux -)
```

API Summary

Jython Syntax

See example below.

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

AuxiliaryContext **auxiliary** [IN, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

AuxiliaryContext auxiliary [IN, OPTIONAL, default=no default value]

When present the info is copied from the UplinkProduct inside this AuxiliaryContext.

History

- 2009-11-10 DK

1.65. DoVelocityCorrection

Full Name:	herschel.hifi.pipeline.generic.DoVelocityCorrection
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import DoVelocityCorrection

Description

Corrects the frequency for the velocity of the spacecraft using a relativistic or approach; when transforming to the rest frame of the source various different approaches are available.

By setting the parameter `targetFrame` you can specify what frequency scale to transform to. Once the transformation is done, the meta data field 'freqFrame' is updated accordingly. The available frames that can be transformed to are:

- "hso" or "topocentric"
- "geocentric"
- "ssb" or "barycentric"
- "lsr" or "lsrk"
- "source"

If the 'targetFrame' parameter is not specified it is set to 'lsr' for non SSO or 'source' for SSO's. If the task succeeds the meta data item 'freqFrame' in the timeline product and the spectrum datasets is set with the target frame ("topocentric","geocentric","barycentric","lsrk","source"). At the beginning, the task checks whether the 'freqFrame'-parameter has been set and considers that as the frame the frequency scale needs to be transformed from. In case no such field is found the freqScale of the input is assumed to be the spacecraft (hso or topocentric). Depending on the initial and the target frequency frame, different velocity information is needed. Except for the velocity of the source, all this velocity information is expressed in barycentric coordinates:

- spacecraft velocity
- earth velocity
- LSR velocity

This information is found in different sources:

- in the data (spacecraft velocity) in case no 'aux' and no 'ephem' parameters has been set;
- in the OrbitEphemerisProduct of the auxiliary context (spacecraft velocity) in case the 'aux' parameter is set to `obs.auxiliary`;
- the HCSS build (velocity of earth and LSR)
- in an ephemeris product (all SSO including the spacecraft) if the 'ephem' parameters has been set with a object of type `herschel.share.fltdyn.ephem.Ephemerides`

For SSO's Horizons ephemerides data is also needed for the motion of the source. - Here, an ephemerides object needs to be passed to the task. The velocity of the source (non SSO) may be expressed in different frames and/or different formulas. Actually, it is not necessarily the physical velocity of the source that is specified here but rather the redshift of the source relative to the reference frame. Depending on the formula to express the redshift, the parameter entering there has a different meaning. The available formulas to express the redshift factor:

- optical: `factor = (1+v/c)`
- radio: `factor = 1 / (1-v/c)`
- redshift: `factor = 1+z`
- relativistic: `factor = sqrt((1+v/c)/(1-v/c))`

In all cases, the parameter entering these formulas (v or z) can be passed to the task by setting the parameter 'redshift' or, equivalently, 'velocity_source'. In case no such task parameter is set and the target frame is set to 'source' a suitable parameter is looked up in the timeline product (meta data item 'redshift' or 'vlsr'). The formula to be adopted to translate the redshift parameter into a redshift factor can be specified by setting the task parameter 'redshiftType' (to optical, radio, redshift or relativistic). Again, in case no such parameter is specified a meta data parameter 'redshiftType' is looked up in the timeline product. If no redshift type information is found the default 'optical' is applied. The reference frame the redshift of the source is applied to is specified by setting the parameter 'redshiftFrame'. In case no such parameter is specified a meta data parameter 'frame' is looked up in the timeline product. If no redshift frame is specified the default 'lsr' is used. After such a redshift factor is successfully applied to the spectra the information for building it is included in the meta data.

- 'redshift': the redshift parameter;
- 'frame': the reference frame the redshift is expressed in;
- 'redshiftType': type of redshift formula used to compute from the redshift parameter the redshift;
- 'v_frame': a velocity associated with the redshift parameter computed by referring to the optical definition of the redshift.

The 3d velocity information read from the input data and used to transform the frequency scale is entered in the datasets:

- The velocity of the spacecraft in barycentric coordinates as three columns in the datasets (velocity_hso_k).
- For SSO's, the velocity of the source in barycentric coordinates as three columns in the datasets (velocity_sso_k).

The task can be applied for spectra being expressed on the IF scale or the scale frequency scale. In order to consistently transform the spectra from IF to sky frequencies an adjusted LO frequency is added into the datasets. Actually, the original 'LoFrequency'-column is copied to a new column 'LoFrequency_measured' and the original column is adjusted. The ratio of 'LoFrequency' and 'LoFrequency_meas' corresponds to the multiplicative correction factor applied between the current reference frame the frequencies are expressed at and the reference frame of the spacecraft. For frequency switch obsering modes, the correction factor applies also to the 'LoThrow' column and, to preserve the original throw information, a copy 'LoThrow_measured' is created.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.generic import DoVelocityCorrection
doVelocityCorrection = DoVelocityCorrection()
# apply (relativistic) Doppler shifts for the motion of the S/C relative to
LSR, use the velocity information included therein:
doVelocityCorrection(htp=htp, targetFrame="LSR", aux=obs.auxiliary)
# bring it back to the HSO (spacecraft) frame:
doVelocityCorrection(htp=htp, targetFrame="HSO")
# take the velocity of S/C and earth from an ephemerides object:
# given the two files orbitFile,ephemFile
ep = herschel.share.fltdyn.ephem.Ephemerides(orbitFile,ephemFile)
doVelocityCorrection(htp=htp, ephem=ep)
```

Example 1: From jide:

```
# take the velocity of S/C from an ephemerides object
# and use the velocity_source as the velocity relative to LSR (in km/sec)
doVelocityCorrection(htp=htp, targetFrame="source", ephem=ep, redshift=30,
redshiftFrame="LSR", redshiftType="radio"))
```

API Summary

Jython Syntax
See example below.

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]
String **targetFrame** [IN, OPTIONAL, default='LSR']
Object **ephem** [IN, OPTIONAL, default=no default value]
Double **redshift** [IN, OPTIONAL, default=no default value]
Double **velocity_source** [IN, OPTIONAL, default=no default value]
String **redshiftType** [IN, OPTIONAL, default="radio"]
String **redshiftFrame** [IN, OPTIONAL, default='lsr' or 'source' (for SSOs)]
MapContext **aux** [IN, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

String targetFrame [IN, OPTIONAL, default='LSR']

Specify the target frame you would like to correct for. Possible values are (case-insensitive)

- hso or topocentric: the rest frame of the spacecraft
- geocentric: the rest frame of the earth
- ssb or barycentric: solar system barycenter
- lsr or lsrk: Local standard of rest (J2000) - assumed that SSB moves with 20.0 km/sec towards (18h03m50.29s, +30##00'16.8") wrt LSR
- SOURCE: Rest frame of the source.

Object ephem [IN, OPTIONAL, default=no default value]

Pass an ephemeris object or and OrbitEphemeris-product from which the velocity information can be extracted. Once this parameter is set, the velocity information found in the ephemeris

Object ephem [IN, OPTIONAL, default=no default value]

object is used to compute the Doppler correction and the 3d-velocity included in the datasets (columns 'velocity_x'). If this parameter is not set the task assumes that the required velocity information is already available within the data.

Double redshift [IN, OPTIONAL, default=no default value]

The input parameter for computing the redshift applied when transforming to the rest frame of the source. It is either

- a velocity (in km per second) when using `redshiftType=optical` or `redshiftType=radio`
- a redshift parameter z when using `redshiftType=redshift`

This parameter is expressed in a reference frame given by `redshiftFrame`. This parameter is included in the meta data of the datasets and the timeline product as "redshiftFrame". Note that this parameter (similarly `redshiftType`, `redshiftFrame`) is only applied in observations of non-SSO's.

Double velocity_source [IN, OPTIONAL, default=no default value]

The same as the 'redshift' parameter.

String redshiftType [IN, OPTIONAL, default="radio"]

The formula adopted to relate the redshift and the `redshift` parameter (v or z). You have the following possibilities:

- "optical": redshift factor = 1+v/c (i.e. redshift parameter v)
- "radio": redshift factor = 1/(1-v/c) (i.e. redshift parameter v)
- "redshift": redshift factor = 1+z (i.e. redshift parameter z)

This parameter is included in the meta data of the datasets and the timeline product as "redshiftType".

String redshiftFrame [IN, OPTIONAL, default='lsr' or 'source' (for SSOs)]

The reference frame the redshift parameter is expressed in:

- hso or topocentric
- geocentric
- ssb, ssbc, barycentric or heliocentric (as in HSPOT)
- lsr or lsrk
- source

This parameter is included in the meta data of the datasets and the timeline product as "redshiftFrame".

MapContext aux [IN, OPTIONAL, default=no default value]

An auxiliary context from which the OrbitEphemeris product can be extracted. This is used once the 'ephem' parameter is not set. If velocity data has already been entered into the data this information is overwritten.

1.66. DoVlsr

Full Name:	herschel.hifi.pipeline.level0.DoVlsr
Type:	Java Task -
Import:	from herschel.hifi.pipeline.level0 import DoVlsr

Description

Copies the V_{lsr} and the name of the reference frame from the

UplinkProduct inside the AuxiliaryContext to the MetaData of the HTP and the HSDS inside it. Alternatively it can write a user-provided V-lsr and/or frame in the HTP and HSDS. MetaData names are "VLSR" and "FRAME" resp.

Example

Example 1: From jide:

```
from herschel.hifi.pipeline.level0 import DoVlsr
doVlsr = DoVlsr()
doVlsr( htp=htp, auxiliary=aux -)
doVlsr( htp=htp, vlsr=10.0, frame="LSR" -)
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INOUT, MANDATORY, default=no default value]

AuxiliaryContext **auxiliary** [IN, OPTIONAL, default=no default value]

Double **vlsr** [IN, OPTIONAL, default=no default value]

String **frame** [IN, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=no default value]

The timeline product (observation) to be passed to the module.

AuxiliaryContext auxiliary [IN, OPTIONAL, default=no default value]

When present the info is copied from the UplinkProduct inside this AuxiliaryContext.

Double vlsr [IN, OPTIONAL, default=no default value]

It copies this value when it cannot be found in the AuxiliaryContext (e.g. because it is not provided.)

String frame [IN, OPTIONAL, default=no default value]
It copies this value when it cannot be found in the AuxiliaryContext (e.g. because it is not provided.)

History

- 2009-10-28 DK

1.67. DoWbsBadPixels

Full Name:	herschel.hifi.pipeline.wbs.DoWbsBadPixels
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import DoWbsBadPixels
Category:	HIFI pipeline task

Description

Step3: Task that apply the masking pixel list.Configurable to select/unselect masks.

SpectrumDataset output:bad pixel are masked out Calibration object output:- GUI output (plots): Framework functionality needed: Remarks: This module can be reused between each step to decide which pixel should be used or not. HifiSpectrumDataset request field: Name/position of flags

Examples

Example 1: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
spectra = DoWbsScanCount()(htp=spectra)
#step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask)
#step 2
spectra = DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1)  #step 3
```

Example 2: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra)             #step 1
calPixel=MkWbsBadPixels()(spectra)  #step 2
DoWbsBadPixels()(spectra,calPixel)  #step 3
```

API Summary

Jython Syntax

```
See example below.

<p>
```

Properties

{@link HifiTimelineProduct} **"htp"** [IO, Yes mandatory, default=no default value]
{@link } **"cal"** [IN, Yes mandatory, default=no default value]
{@link Boolean} **"apply"** [IN, No mandatory, default=default Value=true]

API details

Properties

{@link HifiTimelineProduct} "htp" [IO, Yes mandatory, default=no default value]

No default value.

Provides the SpectrumDataSet that will be masked.

{@link } "cal" [IN, Yes mandatory, default=no default value]

no default value .

Provides the PixelList that will be applied to mask the SpectrumDataset

{@link Boolean} "apply" [IN, No mandatory, default=default Value=true]

Provides the commands parameters to change the masked pixels in the SpectrumDataSet = BooleanParameter(false). .

History

- 15-May-2005 AL : Javadoc and help completed.
- parameter renamed for more compatibility with hrs
- 14-Jul-2005 AL : First implementation completed.
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.68. DoWbsDark

Full Name:	herschel.hifi.pipeline.wbs.DoWbsDark
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import DoWbsDark

Description

Step4: perform subtraction of dark values for

all scans in the HifiSpectrumDataset.

DARK1_2 =0; Use pixel 0 (first) for even Dark and pixel 1(second) for odd Dark

DARK3_4=1; Use pixel 2 (third) for even Dark and pixel 3(fourth) for odd Dark

DARK_AVERAGE=2; Use the average of pixels 0 and 2 for even Dark and pixels 1 and 3 for odd Dark

Examples

Example 1: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
hk=AccessPacketTask()(obsid=obsid,apid=1026)
spectra=HifiTimelineProduct(hdf,hk)
spectra = DoWbsScanCount(htp=spectra)       #step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask)
#step 2
spectra = DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1 -)  #step 3
dd=DoWbsDark()
#step 4
spectra  = dd(htp=spectra,darkKind=DoWbsDark.DARK1_2)
```

Example 2: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra)             #step 1
calPixel=MkWbsBadPixels()(spectra)  #step 2
MkWbsBadPixels()(spectra,calPixel)  #step 3
DoWbsDark()(spectra)                #step 4
```

API Summary

Properties
{@link HifiTimelineProduct} **"htp"** [IO, Yes mandatory, default=no default value]
{@link Integer} **"darkKind"** [IN, No mandatory, default=no default value]
{@link Boolean} **"usePixel"** [IN, No mandatory, default=no default value]

API details

Properties

{@link HifiTimelineProduct} "htp" [IO, Yes mandatory, default=no default value]

No default value.

Provides the SpectrumDataSet that will be corrected for the dark. *

{@link Integer} "darkKind" [IN, No mandatory, default=no default value]

yes default value = DARK1_2 Provides the Parameters used to decide which kind of dark subtraction is applied.

{@link Boolean} "usePixel" [IN, No mandatory, default=no default value]

yes default value = false Decide that in case of differences between the first Pixels of a CCD and the relative Dark the one to be used are the pixel (usePixel=true) or the dark (usePixel=false)

History

- 15-May-2005 AL : Javadoc and help completed.
- parameter renamed for more compatibility with hrs
- 14-Jul-2005 AL : First implementation completed.
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.69. DoWbsFreq

Full Name:	herschel.hifi.pipeline.wbs.DoWbsFreq
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import DoWbsFreq
Category:	HIFI pipeline task

Description

Step9: create frequency table for each scan.

HifiSpectrumDataset request field: Name/position of Overlapping flags

Examples

Example 1: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf) spectra =
DoWbsScanCount()(htp=spectra) #step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask) #step 2
spectra = DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1 -) #step 3
dd=DoWbsDark() #step 4
spectra = dd(htp=spectra,darkKind=LongParameter(DoWbsDark.DARK1_2))
spectra = DoWbsNonlin()(htp=spectra,cal=MyTablePolyn) #step 5
mkz=MkWbsZero() #step 6
zeroCal=mkz(htp=spectra,interp=LongParameter(zeroCal.NONE))
zerocheck=mkz.zeroCheck
spectra =DoWbsZero()(htp=spectra,cal=zeroCal) #step 7
mkf=MkWbsFreq() #step 8
freqCal=mkf(htp=spectra,interp="PREVIOUS",COMB_FIRST_LINE_POSITION=160,
COMB_LINES_STEP=175,
     COMB_THRESHOLD=500)
 combCheck=mkf.combCheck #equivalent to: combCheck=freqCal.check
 DoWbsFreq()(htp=spectra,cal=freqCal) #step 9
```

Example 2: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra) #step 1
calPixel=MkWbsBadPixels()(spectra) #step 2
MkWbsBadPixels()(spectra,calPixel) #step 3
DoWbsDark()(spectra) #step 4
DoWbsNonlin()(spectra) #step 5
zeroCal=MkWbsZero()(spectra) #step 6
DoWbsZero()(spectra,zeroCal) #step 7
freqCal=MkWbsFreq(spectra) #step 8
DoWbsFreq()(spectra,freqCal) #step 9
```

API Summary

Jython Syntax

```
See example below.
<p>
```

Properties

{@link HifiTimelineProduct} **"htp"** [IO, Yes mandatory, default=No]

Properties
{@link CalWbsFreq} **"cal"** [IN, Yes mandatory, default=No default]
No mandatory **"model"** **IN** [{@link CalWbsFreq}, default value "column", default=no default value]

API details

Properties

{@link HifiTimelineProduct} "htp" [IO, Yes mandatory, default=No]

default value.

Provides the (Science or technical) SpectrumDataSet where the frequency calibration has te be applied.

{@link CalWbsFreq} "cal" [IN, Yes mandatory, default=No default]

value.

Provides the FrequencyCalibration to be applied to the SpectrumDataSet .

If it is a null the model inserted in the spectrum is a Wbs2DFreqModel() with no input parameters.

No mandatory "model" IN [{@link CalWbsFreq}, default value "column", default=no default value]

of this Task permits to choose between: - if "model" = "model" ==> creation of the Wbs2DFreqModel , - if "model" = "column" ==> creation of wave columns with the frequency in them. If the wave columns are chosen, this tasks will create wave columns named "frequency"

History

- 15-May-2005 AL : Javadoc and help completed parameter renamed for
- more compatibility with hrs 14-Jul-2005 AL : First implementation
- completed.
- 15-Jun-2006 AL : cal not more mandatory
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.70. DoWbsNonlin

Full Name:	herschel.hifi.pipeline.wbs.DoWbsNonlin
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import DoWbsNonlin
Category:	HIFI pipeline task

Description

Step5: perform nonlinearity correction for even and odd WBS pixels.

Examples

Example 1: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
spectra =DoWbsScanCount()(htp=spectra) #step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask) #step 2
spectra =DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1 -) #step 3
spectra =DoWbsDark()(htp=spectra,darkKind=DoWbsDark.DARK1_2)  #step 4
spectra =DoWbsNonlin()(htp=spectra,cal=DefaultValues.getLinearCoeff())#step 5
```

Example 2: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra) #step 1
calPixel=MkWbsBadPixels()(spectra) #step 2
DoWbsBadPixels()(spectra,calPixel) #step 3
DoWbsDark()(spectra)   #step 4
DoWbsNonlin()(spectra) #step 5
```

API Summary

Jython Syntax
`See example below.`

Properties
`{@link HifiTimelineProduct} "htp" [IO, Yes mandatory, default=No]`
`{@link CalWbsLinearCoeff} "cal" [IN, No mandatory, default=yes]`

API details

Properties

`{@link HifiTimelineProduct} "htp" [IO, Yes mandatory, default=No]`

default value.

Provides the SpectrumDataSet that will be corrected for the nonlinearity. *

{@link CalWbsLinearCoeff} "cal" [IN, No mandatory, default=yes]
default value=CalWbsLinearCoeff() Provides the Parameters of the polynomials per band and pixel type for the pixel non linearity corrections.

History

- 15-May-2005 AL : Javadoc and help completed. parameter renamed for
- more compatibility with hrs 14-Jul-2005 AL : First implementation
- completed.
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.71. DoWbsScanCount

Full Name:	herschel.hifi.pipeline.wbs.DoWbsScanCount
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import DoWbsScanCount
Category:	HIFI pipeline task

Description

Step1: Task for the correction of scan Count and for the

integration time. - `HifiWbsDataFrame`

Examples

Example 1: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
hk=AccessPacketTask()(obsid=obsid,apid=1026)
spectra=HifiTimelineProduct(hdf,hk)
spectra = DoWbsScanCount(htp=spectra)
```

Example 2: From jide:

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=HifiTimelineProduct(hdf) #step 0
DoWbsScanCount()(spectra)   #step 1  renamed
```

API Summary

Jython Syntax
See example below. <p>

Property
{@link HifiTimelineProduct}[] **"htp"** [IO, Yes mandatory, default=no default value]

API details

Property

{@link HifiTimelineProduct}[] "htp" [IO, Yes mandatory, default=no default value]

No default value.

Provide the scan count correction on the input HifiTimelineProduct

History

- 15-May-2005 AL : Javadoc and help completed

- parameter renamed for more compatibility with hrs
- 14-Jul-2005 AL : Implementation completed
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.72. DoWbsSubbands

Full Name:	herschel.hifi.pipeline.wbs.DoWbsSubbands
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import DoWbsSubbands
Category:	HIFI pipeline task

Description

Step11: perform the splitInCcd() in all HifiSpectrumDataset contained in the HifiTimelineProduct.

Example

Example 1: From jide:from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra) #step 1
calPixel=MkWbsBadPixels()(spectra) #step 2
MkWbsBadPixels()(spectra,calPixel) #step 3
DoWbsDark()(spectra) #step 4
DoWbsNonlin()(spectra) #step 5
zeroCal=MkWbsZero()(spectra) #step 6
DoWbsZero()(spectra,zeroCal) #step 7
freqCal=MkWbsFreq(spectra) #step 8
DoWbsFreq()(spectra,freqCal) #step 9
att=MkWbsFluxAtten()(spectra) #step 10
DoWbsSubbands()(spectra)
```

API Summary

Jython Syntax

See example below.

<p>

Property

{@link HifiTimelineProduct} **"htp"** [IO, Yes mandatory, default=no default value]

API details

Property

{@link HifiTimelineProduct} "htp" [IO, Yes mandatory, default=no default value]

No default value.

1.73. DoWbsZero

Full Name:	herschel.hifi.pipeline.wbs.DoWbsZero
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import DoWbsZero
Category:	HIFI pipeline task

Description

Step7: Subtract appropriate zero spectra for each time step. If the zeros are

subtracted a new MetaData "isZeroApplied" with BooleanParameter =true will be add to the WbsSpectrumDataset. If the BooleanParameter is present and true the zero is not subtracted.

Examples

Example 1: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
spectra = DoWbsScanCount()(htp=spectra)
#step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask)
#step 2
spectra = DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1 -)  #step 3
dd=DoWbsDark()
#step 4
spectra  = dd(htp=spectra,darkKind=0)
spectra = DoWbsNonlin()(htp=spectra,cal=MyTablePolyn)                #step 5
zeroCal=MkWbsZero()(htp=spectra,interp="PREVIOUS")
zerocheck=zeroCal.check
spectra = DoWbsZero()(htp=spectra,cal=zeroCal)
#step 7
```

Example 2: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra)             #step 1
calPixel=MkWbsBadPixels()(spectra)  #step 2
MkWbsBadPixels()(spectra,calPixel)  #step 3
DoWbsDark()(spectra)                #step 4
DoWbsNonlin()(spectra)              #step 5
zeroCal=MkWbsZero()(spectra)        #step 6
DoWbsZero()(spectra,zeroCal)        #step 7
```

API Summary

Jython Syntax

See example below.

<p>

Properties

{@link HifiTimelineProduct} **"htp"** [IO, Yes mandatory, default=no default value]

Properties
{@link CalWbsZero} **"cal"** [IN, Yes mandatory, default=no default value]

API details

Properties

{@link HifiTimelineProduct} "htp" [IO, Yes mandatory, default=no default value]
No default value. Provides the (Science or technical) SpectrumDataSet where the zeros calibration has to be applied.

{@link CalWbsZero} "cal" [IN, Yes mandatory, default=no default value]
No default value. Provides the ZeroCalibration to be applied to the SpectrumDataSet . If empty ZeroCalibration is provided the spectrum is unchanged.

History

- 15-May-2005 AL : Javadoc and help completed
- parameter renamed for more compatibility with hrs
- 14-Jul-2005 AL : First implementation completed.
- 15-Jun-2006 AL : cal not more mandatory
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.74. FreqRanges

Full Name:	herschel.hifi.pipeline.generic.FreqRanges
Type:	Java Class -
Import:	from herschel.hifi.pipeline.generic import FreqRanges

Description

Product containing the measures for frequency drifts potentially occurring during an observation.

For each group ob datasets in the original timeline product (defined by consistent LO settings) a {@link TableDataset} is constructed which contains the per channel frequencies for the comb measurements and the drift measure computed for sub-sequent frequency grids.

The drift measure is calculated by

- first, computing for each subband the average difference between corresponding channel-frequencies of the sub-sequent comb measurements and
- second, taking the maximum over the absolute values of the per subband measures computed in the first step.
- Finally, this distance is divided by teh distiance in observation time.

The drift measure is reported in units Hz / sec.

1.75. GriddingTask

Full Name:	herschel.hifi.dp.otf.GriddingTask
Type:	Java Task -
Import:	from herschel.hifi.dp.otf import GriddingTask
Category:	HIFI

Description

produces a cube of images from a HifiTimelineProduct,

makes a cube for each given subband (segment). The images are produced by performing two dimensional gridding of a number of spectra irregularly distributed over some area of the sky observed in a mapping mode. The spectra are read from a number of SpectrumContainers within an SpectrumContainerBox or from a single SpectrumContainer.

This task computes a regular grid that covers the region of the sky where the spectra have been observed.

For each pixel of that grid, the task computes a normalized convolution of those spectra which fall in the convolution kernel around such pixel.

The user can choose between assigning the same weight to every spectrum or using the weights computed by the generic part of the HIFI pipeline (i.e. use the weights columns of the HifiSpectrumDatasets).

Besides, the user has to choose between several type of filter functions to compute the contribution of each spectrum to each pixel, based on the distance of the spectrum to the centre of the kernel.

The user can also use an smoothing factor and provide some other parameters, e.g. to control the geometry of the regular grid and the range of channels to use to create the cube (i.e. to clip the wave range).

Examples

Example 1: make cube for first subband

```
subband = 1
cube = griddingTask(container=spectrumContainer, segments=Int1d([subband]),
beam=Double1d([25.0]))
```

Example 2: make cubes for all the subbands, then Display the first one

```
cube = griddingTask(containerBox=spectrumContainerBox, beam=Double1d([25.0]))
cubes = griddingTask.cubes
cubes_count = len(cubes)
Display(cube)  # displays cube[0]
```

Example 3: make cubes for the subbands 1 and 3, then Display the first one

```
cube = griddingTask(containerBox=spectrumContainerBox, segments=Int1d([1,3]),
beam=Double1d([25.0]))
cubes = griddingTask.cubes
cubes_count = len(cubes)
cube_subband_1 = cubes[0]
cube_subband_3 = cubes[1]
Display(cube)
```

API Summary

Jython Syntax
`cubes = griddingTask(container=spectrumContainer [, segments=Int1d([i,j, ..., n]), ...])`

Properties
SpectrumContainer **container** [INPUT, OPTIONAL, default=no default value]
SpectrumContainerBox **containerBox** [INPUT, OPTIONAL, default=no default value]
Int1d **subbands** [INPUT, OPTIONAL, default=no default value]
Double **"beamSize"** [INPUT, OPTIONAL, default=no default value]
String **weightMode** [INPUT, OPTIONAL, default="equal"]
Int2d **channels** [INPUT, OPTIONAL, default=empty by default]
Double1d **mapSize** [INPUT, OPTIONAL, default=Int2d(2,0) i.e. autocompute optimal]
Double1d **refPixel** [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]
Double1d **refPixelCoordinates** [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]
Double1d **pixelSize** [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]
Double1d **smoothFactor** [INPUT, OPTIONAL, default=Double1d(2, 1.0), i.e. autocompute]
Integer **filterType** [INPUT, OPTIONAL, default=2]
Double1d[] **filterParams** [INPUT, OPTIONAL, default=no default value]
Cube **cube** [OUTPUT, OPTIONAL, default=no default value]
Cube[] **cubes** [OUTPUT, OPTIONAL, default=no default value]
Double1d **xPoints** [OUTPUT, OPTIONAL, default=no default value]
Double1d **yPoints** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

SpectrumContainer container [INPUT, OPTIONAL, default=no default value]

container of the spectra to be convolved into a regular grid to produce a cube of images.

SpectrumContainerBox containerBox [INPUT, OPTIONAL, default=no default value]

set of containers of spectra to be convolved into a regular grid to produce a cube of images.

Int1d subbands [INPUT, OPTIONAL, default=no default value]

specifies the subbands to be used, to produce a cube for each given subband of the input HifiTimelineProduct.

Int1d subbands [INPUT, OPTIONAL, default=no default value]

necessary only when the spectra in the given HifiTimelineProduct have several subbands.

Double "beamSize" [INPUT, OPTIONAL, default=no default value]

No default value.

Provides the antenna beam width, **in arcseconds**.

String weightMode [INPUT, OPTIONAL, default="equal"]

specifies how to assign spectral weights;

every channel of every spectrum receives its own weight.

The weighting strategies are:

- "equal": all channels of all spectra have the same weight.
- "selection": takes the "weight" column of the spectra.

Please note that for each spectrum, each channel can have a different weight so the weighting is done channel-wise.

Int2d channels [INPUT, OPTIONAL, default=empty by default]

by default, in this case all channels are selected for each segment (subband)

This parameter can be used to "crop" a part of spectral range.

For each segment, this Int2d has a row with two elements, the first element the start channel, and the second element specifies the last channel, i.e. the start and the end of the range of channels to be read per segment

Double1d mapSize [INPUT, OPTIONAL, default=Int2d(2,0) i.e. autocompute optimal]

this parameter may be used to specify the size of the map to be generated; the size is given along the x and y axes, in **arc seconds**.

When both sizes are equal, a Double1d with a single element can be provided.

Otherwise, this parameter array must contain two elements:

- width of the map, size along the x axis
- height of the map, size along the y axis

By default, this parameter is set to zero, in order to let algorithm choose the optimal map dimensions.

Double1d refPixel [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

specifies which is the reference pixel (in both axes), whose offset with regards to the projection centre is given by the (additional) parameter refPixelOffset

Double1d refPixel [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

By default, this parameter is set to zero, to let the algorithm find the pixel coordinates of the point to which the spectra offsets are referred.

If both x- and y-axis values of the reference pixel are equal, a Double1d array with a single element can be provided.

Otherwise, this parameter must have two elements:

- first element of the Double1d array specifies the value of the reference pixel in the x axis,
- second element of the Double1d array specifies the value of the reference pixel in the y axis

This reference pixel is referred to the bottom most, left most pixel of the regular grid that covers the whole area to be mapped.

Please note that the referencePixel and the offset value at the reference pixel (aka refPixelOffsetValue) are correlated, the user cannot specify an arbitrary value for each of them.

Double1d refPixelCoordinates [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

specifies the coordinates of the reference pixel, in the x and y axes.

By default, this parameter is set to zero to let algorithm compute a suitable reference pixel.

If provided, this parameter must have two elements:

- first element: coordinate of the reference pixel in the x axis (e.g. longitude of the reference pixel),
- second element: coordinate of the reference pixel in the y axis (e.g. latitude of the reference pixel)

Double1d pixelSize [INPUT, OPTIONAL, default=Double1d(2, 0.), i.e. autocompute]

defines the size of the pixels in the x and y axes, measured in arc seconds per pixel.

By default this parameter is set to zero to let algorithm compute the optimal size.

If both x- and y-axis values are equal, a Double1d array with a single parameter can be chosen.

Otherwise, this parameter must have two elements:

- size along the x-axis in the first element,
- size along the y-axis in the second one.

Double1d smoothFactor [INPUT, OPTIONAL, default=Double1d(2, 1.0), i.e. autocompute]

deprecated parameter (usage not recommended);

Double1d smoothFactor [INPUT, OPTIONAL, default=Double1d(2, 1.0), i.e. autocompute]

provides a smooth factor, increasing (broadening) the area of influence (kernel size) used to compute each pixel.

Integer filterType [INPUT, OPTIONAL, default=2]

specifies the type of convolution filter, by default an exponential (gaussian) function.

Allowed values:

- BoxCar => 1
- Exponential (Gaussian) => 2

Visit the herschel.hifi.dp.otf.filter package documentation for details about the available filters.

Double1d[] filterParams [INPUT, OPTIONAL, default=no default value]

provides the numeric parameters for that characterize the filter, such as width and position.

This input parameter is an array that contains two Double1d arrays.

- The first element of this input array i.e. the first Double1d provides the parameters for the filter function along the x-axis,
- The second element of this input array i.e. the second Double1d provides the parameters for the filter function along the y-axis

For example, if the filter type is gaussian, the filterparam[0].get(0) contains the kernel length (in pixels) along the x-axis , and the kernel beam width in x-axis(in pixels).

As a second example, if the filter type is box then filterparam[0] contains contains just a Double1d with a single-element, the kernel length (in pixels).

Cube cube [OUTPUT, OPTIONAL, default=no default value]

No default value.

First cube produced by this task, if several segments where processed, or the only cube produced if only one segment was specified

Cube[] cubes [OUTPUT, OPTIONAL, default=no default value]

No default value.

Array of cubes produced by this task, one per (specified) subband of the HifiTimelineProduct.

Double1d xPoints [OUTPUT, OPTIONAL, default=no default value]

No default value.

x-coordinates of points that define the regular grid.

Double1d yPoints [OUTPUT, OPTIONAL, default=no default value]
No default value. y-coordinates of points that define the regular grid

History

- 04-Jun-2009 First version. Based on DoCube.

1.76. HiClassTask

Full Name:	herschel.hifi.dp.tools.HiClassTask
Type:	Jython Task -
Import:	from herschel.hifi.dp.tools import HiClassTask
Category:	task

Description

Export HIFI spectra to a FITS file that CLASS can read.

This task is designed to export HifiSpectrumDatasets (at least level 0.5: having gone through the instrument pipeline) to a FITS file that CLASS can understand. This task is a wrapper around the HiClass object defined in herschel/hifi/dp/tools/hiclass_tools.py. For a complete documentation of the HiClass object, please type print herschel.hifi.dp.tools.hiclass_tools.__doc__

Examples

Example 1: Simply export one dataset to a FITS file:

```
HiClassTask()(dataset = myspectra, fileName = -'myspectra.fits')
```

Example 2: Simply export one HIFI timeline product to a FITS file:

```
HiClassTask()(product = myhtp, fileName = -'myhtp.fits')
```

Example 3: Export an entire HifiTimelineProduct dataset by dataset:

```
hiclasstask = HiClassTask()
hiclasstask.blankingValue = --9999 # Set it once and for all.
hiclassobj = None
dataset_iterator = htp.iterator() # To loop over the datasets.
while dataset_iterator.hasNext():
    dataset = dataset_iterator.next()
    # Here, you can do some processing on the dataset,
    # like stitching the subbands, or setting to Double.NaN
    # the channels flagged as spurs.  And then you continue:
    hiclasstask.hiClassObj = hiclassobj # Re-inject the previous HiClass
object.
    hiclasstask.dataset = dataset # We want to add this dataset.
    hiclassobj = hiclasstask() # Add it.
# Export to fits file.
hiclasstask.dataset = None # We're not adding datasets anymore.
hiclasstask.hiClassObj = hiclassobj # We're exporting this result.
hiclasstask.fileName = -'myhtp.fits' # We're exporting to this file.
hiclasstask() # Export the HiClass product into the FITS file.
```

Example 4: See what does the HiClass Product look like.

```
# after the previous example, enter:
hiclassobj = hiclasstask()
hiclassprod = hiclassobj.Prod
# now, look at it with the Dataset Explorer.
# It is the product exported by FitsArchive.
```

API Summary

Jython Syntax

```
HiClassTask()(ds, fileName = 'myfits.fits')
```

Properties
HifiSpectrumDataset **dataset** [INPUT, OPTIONAL, default=None]
Product **product** [INPUT, OPTIONAL, default=None]
Integer **blankingValue** [INPUT, OPTIONAL, default=-1000]
String **fileName** [INPUT, OPTIONAL, default='']
Double **raNominal** [INPUT, OPTIONAL, default=NaN]
Double **decNominal** [INPUT, OPTIONAL, default=NaN]
Double **veloSource** [INPUT, OPTIONAL, default=NaN]
String **specsys** [INPUT, OPTIONAL, default='']
OUTPUT **hiClassObj** [INPUT, , default=OPTIONAL]

API details

Properties

HifiSpectrumDataset dataset [INPUT, OPTIONAL, default=None]

A dataset containing the HIFI spectra that you wish to export. If left to its default value None, then no dataset will be added to the HiClass object (see task parameter hiClassObj). Please look at the example to see when it can be useful to set this task parameter to None.

Product product [INPUT, OPTIONAL, default=None]

Like dataset, but for products. The task parameters datasets and product must NOT be set at the same time, otherwise a SignatureException is raised when the task is executed (this is checked in the preamble).

Integer blankingValue [INPUT, OPTIONAL, default=-1000]

NaNs in the flux columns will be replaced by this value. That is the only way to blank channels in CLASS as CLASS does not handle flags or NaNs. Note: the value of this parameter is not only stored in the task but also in the HiClass object. Therefore, even if the task 'forgets' the value of this parameter, you do not need to set it again if you are reusing the same HiClass object.

String fileName [INPUT, OPTIONAL, default='']

The name of the FITS file (path included) that you with to generate. If left empty (''), then no file is created.

Double raNominal [INPUT, OPTIONAL, default=NaN]

raNominal and decNominal allow the user to set the 'central' position of the observation he/she wishes to export. All the positions offsets (well known concept for CLASS users) will be calculated from the point (raNominal ; decNominal). This is of primordial importance for maps. If raNominal or decNominal is not provided, then HiClassTask will try to find it in the dataset or product that you provide. Today (October 22nd 2009), only the ObservationContext contains this information stored in its metadata parameters raNominal and decNominal (same names, it makes things easier). They are defined in the AOR by the proposer. This position may never be pointed at. For example, the proposer can set raNominal and decNominal to the center of a supernova but actually ask for the telescope to point at a shock quite far from this center and want all the offsets to be relative to the center of the supernova. The HifiTimelineProduct is supposed to contain a copy of the ObservationContext nominal ra and dec but it does not yet, as covered by the SPR HIFI-2282. However, another set of coordinates is available at that level and can be used to have the same origin for all the datasets contained in the HTP. These co-

Double raNominal [INPUT, OPTIONAL, default=NaN]

ordinates are the average coordinates of the entire observation and are stored in the metadata parameters "ra" and "dec". It is not the nominal pointing but it is an actual pointing that all datasets can share: not as desirable, but better than nothing. So if the task parameters raNominal and decNominal are left to NaN, then the HIFI to CLASS exporter will look into the products for suitable metadata parameters to use. If absolutely nothing is found then a warning is issued and each dataset will have for origin its averaged position. This can problem for maps. That is why if you use this task with maps at a dataset level you are strongly encouraged to fill the task parameters raNominal and decNominal.

Double decNominal [INPUT, OPTIONAL, default=NaN]

See raNominal.

Double veloSource [INPUT, OPTIONAL, default=NaN]

The velocity of the source in the reference frame in which the frequencies of are expressed in the input data. If left to its default value NaN, then the velocity will be read from the input data. This parameter is presented to you in order to override the velocity provided in the proposal when this one is wrong.

String specsys [INPUT, OPTIONAL, default='']

If left to its default value (empty string), then HiClass will try to find in the datasets the reference frame in which the frequencies are expressed. Setting this parameter to something else than '' will override whatever reference frame the datasets may refer to. This describes the frame in which the original data is provided. In any case, the frame used at the end in the CLASS file is the one of the source. The name of this parameter, 'specsys', comes from the paper "Representations of spectral coordinates in FITS" by Greisen et al, 2005. This defines the following legal values for this parameter : * TOPOCENT: topocentric * GEOCENTR: geocentric * BARYCENT: barycentric * HELIOCEN: heliocentric * LRSK : local standard of rest (kinematic) * LSRD : local standard of rest (dynamic) * GALACTOC: galactocentric * LOCALGRP: local group * CMBDIPOL: cosmic microvave background dipole * SOURCE : source rest frame Since HCSS is not limited to 8 uppercase characters, we chose these legal values for this parameter: * 'topocentric' * 'geocentric' * 'barycentric' * 'heliocentric' * 'LSRk' * 'LSRd' * 'galactocentric' * 'localGrp' * 'CMBdipole' * 'source' From these, only two are recognized by HiClass for now: * 'topocentric' : the frequencies are expressed in the the satellite frame. The satellite velocity correction was not applied. * 'LSRk' : the frequencies are in the local standard of rest. They have been corrected from the effect of the satellite velocity. There is still no correction of the source velocity.

OUTPUT hiClassObj [INPUT, , default=OPTIONAL]

A HiClass object. It contains a Product which, when exported to FITS, can be understood by CLASS. It also contains methods to insert datasets into that Product (quite a lot of processing and reshaping is required). You don't have to know much about that object to be able to use it. One HiClass object corresponds to one FITS file. You can add as many datasets as you want to this HiClass object before exporting it as a FITS file. If you leave the task parameter hiClass to its default value None, then a HiClass object will be created for you. And you usually don't have to touch it after that. Please look at the examples for a practical use of this object. Complete documentation of the HiClass object : print herschel.hifi.dp.tools.hiclass_tools.__doc__

History

- $Log: HiClassTask.py,v $
- Revision 1.10 2010/02/11 13:49:17 delforge
- Checks the validity of specsys in the preamble.

- Revision 1.9 2010/02/10 16:51:21 delforge
- HiClass now produces the signal and image frequencies in the reference
- frame of the source (rest frequencies) which is the only one CLASS is
- supposed to deal with.
- Two new input parameters were introduced: 'veloSource' and
- 'specsys'. They are used to override the velocity of the source and
- the reference frame in which the frequencies are expressed in the
- input dataset. Useful when the proposal or the pipeline is lacking
- something.
- Revision 1.8 2010/02/03 16:43:31 delforge
- * blanking value set to -1000 instead of 1000
- * better exception handling around the code loading the chopper table
- * exports a tsys value for each spectrum
- * specify LSR as a referential for velocity
- Revision 1.7 2010/02/02 10:43:51 hsclib
- LGPL
- Revision 1.6 2010/01/26 14:27:03 delforge
- Reformatted the documentation paragraphs.
- Revision 1.5 2009/12/07 10:18:13 delforge
- Implements SPR HIFI-3173: Typo in documentation corrected.
- Revision 1.4 2009/10/27 09:46:45 delforge
- Implements SPR HIFI-3029: When exporting an HTP or an
- ObservationContext (new feature btw), HiClass will try to find a
- common reference position for all the datasets. This comes as a
- workaround a pipeline problem: the pipeline doesn't fill the raNominal
- and decNominal of the datasets yet, so the information has to be found
- somewhere else. The user can also set raNominal and decNominal
- himself now by setting the right task parameters (HiClassTask) or
- configuration items (HiClass).
- Revision 1.3 2009/10/09 11:34:06 delforge
- Implements SCR HIFI-2983: HiClassTask is now visible from Hipe. In
- order for this to really happen, there is a new version of all.py in

- herschel.hifi.dp to be committed as well.
- Revision 1.2 2009/10/05 13:51:21 delforge
- Implements SCR HIFI-2975: The task can export products as well as
- datasets and allow the configuration of the HiClass object.
- Revision 1.1 2009/09/11 14:58:33 delforge
- First version in hifi.dp.tools.
- 10-10-2008 BD: Creation.

1.77. HifiFitFringe

Full Name:	herschel.hifi.dp.tools.FitHifiFringe
Alias:	HifiFitFringe
Type:	Jython Task -
Import:	from herschel.hifi.dp.tools import FitHifiFringe

1.78. HifiPipelineTask

Full Name:	herschel.hifi.pipeline.HifiPipelineTask
Alias:	HifiPipelineTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline import HifiPipelineTask

Description

Task to wrap pipeline algorithms.

You will find the gui for this task (hifiPipeline) under applicable tasks of a ObservationContext. Double-click on it to open it. In the basic mode it allows you to re-run the pipeline for each of the backends individually and from any (present) level to some higher level. For the "fromLevel=-1" (i.e reprocess the complete Observation from scratch) you need access to the telemetry database which is generally not available outside the HIFI-ICCs. In the 4 boxes you can select your own algorithms for the different parts of the hifi pipeline. This already requires quite some expert knowledge about the instrument. In the expert mode even more (obscure) options are available. When starting from scratch you can select an obsid and a database. Remember however that in this case you need to have access to the telemetry databases at the HIFI-ICC. The most usefull option is maybe the hifical where you can reprocess the data using a new calibration tree. See also the examples below for commandline usage of the hifi pipeline.

Example

Example 1: PipelineTask

```
<pre>
from  herschel.hifi.pipeline import HifiPipelineTask
# Example 1: Having an existing observation context called obs you can
reprocess it.
# Note that all level 0, calibration and other products (auxilary etc) are
not replaced.
#
hifiPipeline(obs=obs, fromLevel=0)
#
# Note: you can use the Observation Viewer to inspect the outcoming
observation context
#
#
# Example 2: Reprocess only WBS products from level 1 to level 2
#
hifiPipeline(obs=obs, fromLevel=1, upToLevel=2, apids=["1030","1031"])
#
#
# Example 3: Reprocess to level 1 and re-install the calibration tree. See
the next example
# to learn how to use a different calibration tree.
# Note the level 2 is removed.
#
hifiPipeline(obs=obs, fromLevel=0, upToLevel=1, cal=1)
#
#
# Example 4: like 3 now providing your own palStore
#
myStore = herschel.ia.pal.ProductStorage()
#
# the next are registered to retrieve your calibration data:
# local:
myStore.register( LocalStoreFactory.getStore("hifi_cal_dev") -)
# or, server pool:
import herschel.share.util.Configuration
Configuration.setProperty("hifical", -"hifi_ops_cal_dev@iccdb1.sron.rug.nl")
```

Example 1: PipelineTask

```
myStore.register(DbPool.getInstance("hifical"))
#
hifiPipeline( obs=obs, fromLevel=0, upToLevel=1, cal=1, palStore = myStore -)
#
#
# Example 5: Reprocess only from level 1 to level 2, keeping the products in
the level 0.5 context.
#
hifiPipeline( obs=obs, fromLevel=0, removeLevel0_5=0 -)
#
#
# Example 6, like ex. 1 now using your own algorithm for wbs, hrs, level1 and/
or level2:
# one can look -/ edit -/ use the algo"s by loading the following scripts
into jide:
# The WBS pipeline algo is found in the {build_root}/lib/herschel/hifi/
pipeline/wbs/WbsPipelineAlgo.py
# The HRS pipeline algo is found in the {build_root}/lib/herschel/hifi/
pipeline/hrs/HrsPipelineAlgo.py
# The level1 pipeline algo is found in the {build_root}/lib/herschel/hifi/
pipeline/generic/Level1PipelineAlgo.py
# The level2 pipeline algo is found in the {build_root}/lib/herschel/hifi/
pipeline/generic/Level2PipelineAlgo.py
#
hifiPipeline( obs, wbsAlgo=myWbsAlgo, hrsAlgo=myHrsAlgo,
level1Algo=mylevel1Algo, level2Algo=mylevel2Algo)
#
# Example 7: Reprocess to level 2, repopulate cal and aux, redo the quality
pipeline and save in palStore
#
hifiPipeline( obs=obs, fromLevel=0, cal=1, aux=1, save=1, quality=1 -)
#
#
#
# The following examples need telemetry database access. In general this is
NOT available outside the ICCs.
#
# Example 8, generate an observation context import scratch
# the default db = retrieved import the property: var.database.devel, check
with propgen.
#
obs = hifiPipeline( obsid=1342190191L, db='hifi_icc_ops_1@iccdb1.sron.rug.nl
0 READ' -)
#
# Note: you can use the Observation Viewer to inspect the outcoming
observation context
#
#
# Example 9, like ex. 8, but with different datasets per box
#
obs = hifiPipeline( obsid=1342190191L, db='hifi_icc_ops_1@iccdb1.sron.rug.nl
0 READ', datasetsPerBox=1 -)
#
#
# to browse the products in the pool:
def startProductBrowser(storage):
    import herschel.ia.pal.browser.ProductBrowser
    result = ProductBrowser.browseProduct(storage)
    return result
#
res = startProductBrowser(myStore)
</pre>
```

API Summary

Properties
array(Integer) **apids** [IN, OPTIONAL, default=default=all apids]

Properties
ObservationContext **obs** [IO, OPTIONAL, default=(alternative to obsid)]
Long **obsid** [IN, OPTIONAL, default=No default value]
String **db** [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]
Long **datasetsPerBox** [IN, OPTIONAL, default=default= 100]
boolean **cal** [IN, OPTIONAL, default=default= false]
boolean **aux** [IN, OPTIONAL, default=default= false]
ProductStorage **palStore** [IN, OPTIONAL, default=default= SimplePool:"spg-pipeline"]
boolean **save** [IN, OPTIONAL, default=default= false]
PyFunction **wbsAlgo** [INPUT, OPTIONAL, default=default=wbsPipelineAlgo]
PyFunction **hrsAlgo** [INPUT, OPTIONAL, default=default=hrsPipelineAlgo]
PyFunction **level1Algo** [INPUT, OPTIONAL, default=default=level1PipelineAlgo]
PyFunction **level2Algo** [INPUT, OPTIONAL, default=default=level2PipelineAlgo]
Boolean **removeLevel0_5** [INPUT, OPTIONAL, default=default=true]
Float **fromLevel** [INPUT, OPTIONAL, default=default=0]
Float **upToLevel** [INPUT, OPTIONAL, default=default=2]
boolean **quality** [INPUT, OPTIONAL, default=default=true]
String **obsMode** [IN, OPTIONAL, default=default=]
String **tmVersion** [IN, OPTIONAL, default=default=]
String **execMode** [IN, OPTIONAL, default=default=interactive]
Boolean **reprocessAllLevels** [IN, OPTIONAL, default=default=false.]
Boolean **gui** [IN, OPTIONAL, default=default=False.]

API details

Properties

array(Integer) apids [IN, OPTIONAL, default=default=all apids]

The apid that have to be processed

ObservationContext obs [IO, OPTIONAL, default=(alternative to obsid)]

No default value, null allowed. The Observation Context processed , in is none, level0 or level0_5, out is level1 and level2

Long obsid [IN, OPTIONAL, default=No default value]

Alternative input for the observation to be processed. ObservationContext created on the fly

String db [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]

The database containing the specified obsid.

Long datasetsPerBox [IN, OPTIONAL, default=default= 100]

Defines the number of datasets per box. datasetsPerBox should be > 0. The value 0 will return the old structure (before v3.0). This use is deprecated.

boolean cal [IN, OPTIONAL, default=default= false]

If true repopulate the calibration context from the palStore.

boolean aux [IN, OPTIONAL, default=default= false]

If true repopulate the auxillary context from the palStore.

ProductStorage palStore [IN, OPTIONAL, default=default= SimplePool:"spg-pipeline"]

The store where the result will be stored and where the calibration product can be retrieved.

boolean save [IN, OPTIONAL, default=default= false]

If true save the products in the palStore.

PyFunction wbsAlgo [INPUT, OPTIONAL, default=default=wbsPipelineAlgo]

The algorithm for the wbs pipeline.

PyFunction hrsAlgo [INPUT, OPTIONAL, default=default=hrsPipelineAlgo]

The algorithm for the hrs pipeline.

PyFunction level1Algo [INPUT, OPTIONAL, default=default=level1PipelineAlgo]

The algorithm for the level1 pipeline.

PyFunction level2Algo [INPUT, OPTIONAL, default=default=level2PipelineAlgo]

The algorithm for the level2 pipeline.

Boolean removeLevel0_5 [INPUT, OPTIONAL, default=default=true]

Level 0.5 is removed when valid Level 1 data can be formed.

Float fromLevel [INPUT, OPTIONAL, default=default=0]

The starting point level for the pipeline. If no ObservationContext is passed in the input parameter "obs" the level will be always = -1 Values used: LEVEL_RAW=-1 LEVEL0= 0 LEVEL0_5=0.5 LEVEL1= 1

Float upToLevel [INPUT, OPTIONAL, default=default=2]

The final level of the pipeline processed. Values used: LEVEL0=0 LEVEL0_5=0.5 LEVEL1=1 LEVEL2=2

boolean quality [INPUT, OPTIONAL, default=default=true]

Run the quality pipeline too.

String obsMode [IN, OPTIONAL, default=default=]

Expert option. The observation mode to be used in the pipeline. Overwrite the obsMode present in the ObservationContext. This option is only relevant for ILT data where the obsMode was not always present.

String tmVersion [IN, OPTIONAL, default=default=]

Expert option. The telemetry version to be used in the pipeline. This option is only relevant for ILT data where the tmVersion was not always correct.

String execMode [IN, OPTIONAL, default=default=interactive]

Expert option. The execution mode. Expert option, irrelevant to users.

Boolean reprocessAllLevels [IN, OPTIONAL, default=default=false.]

Deprecated parameter. In case you pass an (adopted) observation context and you want to re-process level 0 and following switch this one to TRUE

Boolean gui [IN, OPTIONAL, default=default=False.]

Deprecated parameter. If the progress bar is displayed or not.

History

- 12-Feb-08 PZ Add JTAGs for this task
- 07-Mar-08 PZ Add update example for this task
- 28-March-2008 PZ make pipelineTask with API for CS
- 17-March-2010 DK update of documentation.

1.79. HifiProduct

Full Name:	herschel.hifi.pipeline.product.HifiProduct
Type:	Java Class -
Import:	from herschel.hifi.pipeline.product import HifiProduct
Category:	Products

Description

HifiProduct is an extension of MapContext<-Product, which contains

datasets identified by number. It contains an iterator over the stored datasets.

Example

Example 1: In Jide:

```
vds = HifiProduct()         # create a HifiProduct
s1 = Spectrum1d()            # create some Datasets
s2 = Spectrum2d()
s3 = ArrayDataset( -)
vds.set( s1 -)            # set s1 at number -"0001"
vds.set( s2 -)            # set s2 at number -"0002"
vds.set( s3, 4 -)            # set s3 at number -"0004"
s4 = vds.get( 1 -)            # s4 equals s1
print vds.getCount()        # yields 3 (3 sets in vds)
print vds.getLastIndex()        # yields 4 (last one is at 4)
it = vds.iterator()         # iterator over the datasets.
while it.hasNext(): print it.next().__class__   # print classes
vds.remove( 2 -)            # leaves sets at 1 and 4
vds.collapse()            # renumbers sets to 1 and 2
vds.remove()            # removes last one, leaves only nr 1
```

Limitations

HifiProduct still **is** a MapContext<-Product and can be addressed as such. If you do so, the special methods of HifiProduct are **not** guarandeed to work.

History

- 19-07-2006 DK, rip-off from NumberedProduct

1.80. HifiTimelineProduct

Full Name:	herschel.hifi.pipeline.product.HifiTimelineProduct
Type:	Java Class -
Import:	from herschel.hifi.pipeline.product import HifiTimelineProduct

Description

A MapContext containing HifiSpectrumDatasets.

HifiTimelineProduct is an extension of HifiProduct. It also contains a summary table, listing the types of the HSDs

Example

Example 1: in Jide:

```
# assume dfs and hkp are lists of DataFrames and pertaining HKpackets
wbs = HifiTimelineProduct( dfs, hkp -)   # construct a HTP, all in memory
print wbs.refs.keySet().size()        # number of references to Products
p1 = wbs.refs["1"].product              # get the first product
p2 = wbs.getProduct( 2 -)            # alternative (easier) to get product
d3 = wbs.get( 3 -)                # the dataset inside product 3
# assume dfreader and hkreader are ProductReaders of DFs and HKpackets,
# and store is a ProductStorage
wbs = HifiTimelineProduct( dfreader, hkreader, store -)
# creates a lightweight HTP where all SDS are written to the store.
print wbs.getTypes()            # types of the SDSs
# If you want to select different types into your
print new CalBbid().getTypes()        # for all available types.
wbs1 = HifiTimelineProduct()
wbs1.setTypes( String1d( ["hot","cold","att","stab","science","other"] -))
wbs1.setSkips( String1d( ["zero","comb"] -))
wbs1.fillProduct( dfs, hkp -)
```

1.81. hkplot

Full Name:	herschel.hifi.dp.dataflow.hkplot
Alias:	hkplot
Type:	Jython Task -
Import:	from herschel.hifi.dp.dataflow import hkplot

Description

This task can be used to plot a stream of house keeping data.

This task can also be run on the command line, see example below.

Example

Example 1: hkplot

```
<pre>
# In a new -.py file add the lines (call it qlaHkPlot.py)
from herschel.hifi.dp.dataflow.hkplot import *
DataFlowManager(hkPlot())
Thread.sleep(315360000)   #one year in seconds
# Then from the terminal window type the following
# jylaunch qlaHkPlot.py
</pre>
```

History

- `2007-03-23 - KE`: Updated with a workaround method to run from command line

1.82. HrsCheckFtTask

Full Name:	herschel.hifi.pipeline.hrs.HrsCheckFtTask
Alias:	HrsCheckFtTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import HrsCheckFtTask
Category:	HRS task

Description

Task which checks the Functional Tests of HRS.

- FT00 = switch-on.
- FT01 = Master Reset.
- FT02 = Sine, SquareM or SquareS Built-in-Test.
- FT04 = IRM attenuators test.
- FT05 = Linearity test.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.hrs import *
task = HrsCheckFtTask()
task.gui = 1
print task.name
print task.result
# Using the command line
checkFT = HrsCheckFtTask()(obsid=268440330, apid=1029,
db="ilt_par_10_prop@iccdb1.sron.rug.nl")
checkFT()
print checkFT.name
print checkFT.result
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs import HrsCheckFtTask
from herschel.hifi.pipeline.hrs import HrsCheckFtGuiTask
# test Sine
test = HrsCheckFtTask()
test.apid = Integer(1028)
test.hkapid = Integer(1026)
test.obsid = Long(269484969L)
test.db = -"ilt_par_10_prop@iccdb2.sron.rug.nl"
test.gui = 1
</code>
```

API Summary

Jython Syntax
`See example below.`

Properties
Long **obsid** [INPUT, OPTIONAL, default=null]
Integer **apid** [INPUT, OPTIONAL, default=null]
String **db** [INPUT, OPTIONAL, default=no default value]
PyArray **df_list** [INPUT, OPTIONAL, default=no default value]
PyArray **hk_list** [INPUT, OPTIONAL, default=no default value]
Boolean **gui** [INPUT, OPTIONAL, default=false]
String1d **name** [OUTPUT, OPTIONAL, default=no default value]
Bool1d **result** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htpCF** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_FT01** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_FT02sqs** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_FT02sqm** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_FT02sine** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_FT04** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_FT05** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htpCF_FT05** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_lin** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_eff** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htp_polarSwitch** [OUTPUT, OPTIONAL, default=no default value]
HifiTimelineProduct **htpCF_polarSwitch** [OUTPUT, OPTIONAL, default=no default value]

API details

Properties

Long obsid [INPUT, OPTIONAL, default=null]

Provides the Observation id.

Integer apid [INPUT, OPTIONAL, default=null]

The apid (1028 or 1029).

String db [INPUT, OPTIONAL, default=no default value]

The name of the database to access (ilt_par_10_prop@iccdb2.sron.rug.nl).

PyArray df_list [INPUT, OPTIONAL, default=no default value]

The array of HRS Dataframes.

PyArray hk_list [INPUT, OPTIONAL, default=no default value]

The array of housekeeping packets.

Boolean gui [INPUT, OPTIONAL, default=false]

Allows to handle this task using a gui (1 = true) or not (0 = false).

String1d name [OUTPUT, OPTIONAL, default=no default value]

The name of the Functional Test (ft00).

Bool1d result [OUTPUT, OPTIONAL, default=no default value]

The result of the Functional Test.

HifiTimelineProduct htp [OUTPUT, OPTIONAL, default=no default value]

The htp with the results of the Functional test.

HifiTimelineProduct htpCF [OUTPUT, OPTIONAL, default=no default value]

The htp with Correlation Functions in it.

HifiTimelineProduct htp_FT01 [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test FT01.

HifiTimelineProduct htp_FT02sqs [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test FT02sqs.

HifiTimelineProduct htp_FT02sqm [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test FT02sqm.

HifiTimelineProduct htp_FT02sine [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test FT02sine.

HifiTimelineProduct htp_FT04 [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test FT04.

HifiTimelineProduct htp_FT05 [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test FT05.

HifiTimelineProduct htpCF_FT05 [OUTPUT, OPTIONAL, default=no default value]

The htp with Correlation Functions of FT05 in it.

HifiTimelineProduct htp_lin [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test lin.

HifiTimelineProduct htp_eff [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test eff.

HifiTimelineProduct htp_polarSwitch [OUTPUT, OPTIONAL, default=no default value]

The htp with the result of the Functional test polarSwitch.

HifiTimelineProduct htpCF_polarSwitch [OUTPUT, OPTIONAL, default=no default value]

The htp with Correlation Functions of polarSwitch in it.

History

- `2006-12-06 - OCJ`: Creation.

1.83. HrsHKViewTask

Full Name:	herschel.hifi.hrs.HrsHKViewTask
Alias:	HrsHKViewTask
Type:	Java Task -
Import:	from herschel.hifi.hrs import HrsHKViewTask
Category:	HRS task

Description

Task which displays the values of all digital and analog HK values of HRS into a JTable. With

the "gui=1" option, a table is displayed with colorized cells and a slider to navigate through the HRS HK Packets. With the "gui=0" option, a boolean value is returned. Remarks: At this moment, some HK packets contain bad values.

Example

Example 1: From jide:

```
from herschel.hifi.hrs import HrsHKViewTask
apid = 1028
hkapid = 1026
obsid = 3221226629L
db = -"sovt2_fm_4_prop@iccdb1.sron.rug.nl 0 READ"
hk = accessPacketTask(obsid = obsid, apid = hkapid, db = db)
HrsHKViewTask()(model="FM", apid=1028, hk_list=hk)
</code>
```

API Summary

Jython Syntax
See example below.

Properties
String **model** [INPUT, OPTIONAL, default="FM"]
Integer **apid** [INPUT, MANDATORY, default=no default value]
PyArray **hk_list** [INPUT, MANDATORY, default=no default value]

API details

Properties

String model [INPUT, OPTIONAL, default="FM"]

Model name to select HRS model : "QM" or "FM".

Integer apid [INPUT, MANDATORY, default=no default value]

APID value for HRS HK selection (1028=HRH, 1029=HRV)

PyArray hk_list [INPUT, MANDATORY, default=no default value]

PyArray of HK packets.

History

- `2005-11-03 - OCJ`: : Creation

1.84. hrsPipelineTask

Full Name:	herschel.hifi.pipeline.hrs.HrsPipelineTask
Alias:	hrsPipelineTask
Type:	Jython Task -
Import:	from herschel.hifi.pipeline.hrs import HrsPipelineTask
Category:	task

Description

Process

Example

Example 1: HrsPipelineTask

```
<pre>
#
# Examples:
# hrsPipelineTask(obs=obs, palStore=palStore)
# or process only one apid:
# hrsPipelineTask(apid=1028, obs=obs, palStore=palStore)
# or process an htp:
# hrsPipelineTask(htp=htp)
# or load df and hk
# htp = hrsPipelineTask(obsid=268516902)
#
```

API Summary

Properties
PyFunction **algo** [INPUT, OPTIONAL, default=default=]
Integer **apid** [INPUT, OPTIONAL, default=No default value]
Long **obsid** [IN, OPTIONAL, default=No default value]
Boolean **step** [INPUT, OPTIONAL, default=No default value]
ObservationContext **obs** [IO, OPTIONAL, default=(alternative to htp)]
HifiTimelineProduct **htp** [IO, OPTIONAL, default=(alternative to obs)]
Not Used **cal** [IN, MANDATORY, default=no default value]
default applies binstruct properties **palStore** [product storage used to grep calibration products from, MANDATORY, default=no default value]
String **db** [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]
Boolean **gui** [IN, OPTIONAL, default=default=False.]

API details

Properties

PyFunction algo [INPUT, OPTIONAL, default=default=]

The algorithm of the pipeline.

Integer apid [INPUT, OPTIONAL, default=No default value]

ApID of observation to be processed. Aspected value: 1028, 1029

Long obsid [IN, OPTIONAL, default=No default value]

Alternative input for the observation to be processed. HifiTimelineProduct ObservationContext created on the fly

Boolean step [INPUT, OPTIONAL, default=No default value]

if the pipeline should be run step by step

ObservationContext obs [IO, OPTIONAL, default=(alternative to htp)]

No default value, null allowed. The Observation Context processed , in input Level 0, in Output Level 1

HifiTimelineProduct htp [IO, OPTIONAL, default=(alternative to obs)]

the HifiTimelineProduct to be processed.

Not Used cal [IN, MANDATORY, default=no default value]

default applies binstruct properties palStore [product storage used to grep calibration products from, MANDATORY, default=no default value]

String db [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]

the database containing the specified obsid.

Boolean gui [IN, OPTIONAL, default=default=False.]

only used for a test progress bar. TBD . If the progress bar is displayed or not.

1.85. HrsQlaTask

Full Name:	herschel.hifi.hrs.qla.HrsQlaTask
Alias:	HrsQlaTask
Type:	Java Task -
Import:	from herschel.hifi.hrs.qla import HrsQlaTask
Category:	HRS task

Description

Task which starts the HRS Quick Look Analysis (QLA).

Task which starts the HRS Quick Look Analysis (QLA).

Example

Example 1: From jide:

```
from herschel.hifi.hrs.task import *
hrsnewqla = HrsQlaTask()()
hrsnewqla.stop()
```

API Summary

Jython Syntax
`See example below.`

Property
`DataFlow hrsnewqla [OUTPUT, OPTIONAL, default=no default value]`

API details

Property

`DataFlow hrsnewqla [OUTPUT, OPTIONAL, default=no default value]`

The current DataFlow which is running.

History

- `2006-11-22 - OCJ`: : Creation.

1.86. IltObsContext

Full Name:	herschel.hifi.pipeline.spg.plugin.IltObsContext
Alias:	IltObsContext
Type:	Jython Task -
Import:	from herschel.hifi.pipeline.spg.plugin import IltObsContext
Category:	task

Description

Retrieves the context in which an ILT observation was carried out.

Extracts the observation and test-execution objects from the database. Generates a table contianing used scripts, logs etc. Print summary data to the screen.

Example

Example 1: context searching for a particular obsid

```
from herschel.hifi.scripts.users.share.IltObsContext import *
# create a context searcher
x = IltObsContext()
# apply it to an obsid to get a result table
y = x(obsid=268461173)
# or apply it to a date (within an observation!!')
y = x(date='17 Feb 2007 17:00:02')
# you can retrieve the last observation object for further manipulation
obs = x.observation
# or retrieve the last test execution object
test = x.testExecution
```

API Summary

Jython Syntax

```
table = IltObsContext(obsid=2415919267)
```

Properties

Long **obsid** [INPUT, OPTIONAL, default=No default value]
String **date** [INPUT, OPTIONAL, default=No default value]
FineTime **finetime** [INPUT, OPTIONAL, default=No default value]
CompositeDataset **result** [OUTPUT, MANDATORY, default=no default value]
TestExecution **testExecution** [OUTPUT, MANDATORY, default=no default value]
Observation **observation** [OUTPUT, MANDATORY, default=no default value]

API details

Properties

Long obsid [INPUT, OPTIONAL, default=No default value]

Obsid of the observation to be retrieved

String date [INPUT, OPTIONAL, default=No default value]
A time within an observation (as for example '26 Jan 2007 14:25:23'), as alternative input to an obsid

FineTime finetime [INPUT, OPTIONAL, default=No default value]
A time within an observation, represented as a FineTime as alternative input to an obsid

CompositeDataset result [OUTPUT, MANDATORY, default=no default value]

TestExecution testExecution [OUTPUT, MANDATORY, default=no default value]

Observation observation [OUTPUT, MANDATORY, default=no default value]

History

- 2007-02-27 - First: documented version
- 2008-05-28 - SPR: 1437 - missing tope log, no mib label information available, problems repaired

1.87. IVviewer

Full Name:	herschel.hifi.fpu.ivscan.task.viewIVcurveScans
Alias:	IVviewer
Type:	Jython Task -
Import:	from herschel.hifi.fpu.ivscan.task import viewIVcurveScans
Category:	task

Description

Viewer of IVcurve Scans

API Summary

Jython Syntax
`IVviewer()(obsid = obsid)`

Property
`Long obsid [Input, Optional, default=0]`

API details

Property

`Long obsid [Input, Optional, default=0]`
ObsID of HK data, negative values trigger an interactive selection

History

- 1-11-2006 Initial version generated from Task description parameters

1.88. level1PipelineTask

Full Name:	herschel.hifi.pipeline.generic.Level1PipelineTask
Alias:	level1PipelineTask
Type:	Jython Task -
Import:	from herschel.hifi.pipeline.generic import Level1PipelineTask
Category:	task

Description

Process

Example

Example 1: Level1PipelineTask

```
<pre>
#
# Examples:
# level1PipelineTask(obs=obs)
# or process only one apid:
# or process an htp:
# level1PipelineTask(htp=htp)
#
</pre>
```

API Summary

Properties
PyFunction **algo** [INPUT, OPTIONAL, default=default=]
Integer **apid** [INPUT, OPTIONAL, default=No default value]
ObservationContext **obs** [IO, OPTIONAL, default=(alternative to htp)]
HifiTimelineProduct **htp** [IO, OPTIONAL, default=(alternative to obs)]
Not yet in use for the default algo applied here **cal** [IN, MANDATORY, default=no default value]
type **palStore** [Not yet in use for the default algo applied here, MANDATORY, default=no default value]
Boolean **gui** [IN, OPTIONAL, default=default=False.]
Boolean **removeLevel05** [IN, OPTIONAL, default=default=True.]

API details

Properties

PyFunction algo [INPUT, OPTIONAL, default=default=]

The algorithm for the level1 pipeline.

Integer apid [INPUT, OPTIONAL, default=No default value]

ApID as available in the observation context to be processed. Aspected value: 1028, 1029, 1030, 1031

ObservationContext obs [IO, OPTIONAL, default=(alternative to htp)]

No default value, null allowed. The Observation Context processed , in input Level 0, in Output Level 1

HifiTimelineProduct htp [IO, OPTIONAL, default=(alternative to obs)]

the HifiTimelineProduct to be processed.

Not yet in use for the default algo applied here cal [IN, MANDATORY, default=no default value]

type palStore [Not yet in use for the default algo applied here, MANDATORY, default=no default value]

Boolean gui [IN, OPTIONAL, default=default=False.]

only used for a test progress bar. TBD . If the progress bar is displayed or not.

Boolean removeLevel05 [IN, OPTIONAL, default=default=True.]

If set to true the level 0.5 is removed once the level 1 pipeline was successful.

1.89. level2PipelineTask

Full Name:	herschel.hifi.pipeline.generic.Level2PipelineTask
Alias:	level2PipelineTask
Type:	Jython Task -
Import:	from herschel.hifi.pipeline.generic import Level2PipelineTask
Category:	task

Description

Process

Example

Example 1: Level2PipelineTask

```
<pre>
#
# Examples:
# level2PipelineTask(obs=obs)
# or process only one apid:
# or process an htp:
# level2PipelineTask(htp=htp)
#
</pre>
```

API Summary

Properties
PyFunction **algo** [INPUT, OPTIONAL, default=default=]
Integer **apid** [INPUT, OPTIONAL, default=No default value]
ObservationContext **obs** [IO, OPTIONAL, default=(alternative to htp)]
HifiTimelineProduct **htp** [IO, OPTIONAL, default=(alternative to obs)]
Not yet in use for the default algo applied here **cal** [IN, MANDATORY, default=no default value]
type **palStore** [Not yet in use for the default algo applied here, MANDATORY, default=no default value]
String **db** [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]
String **sideband** [IN, OPTIONAL, default=default=both]
MapContext **result** [OUT, OPTIONAL, default=default=None]
Boolean **gui** [IN, OPTIONAL, default=default=False.]

API details

Properties

PyFunction algo [INPUT, OPTIONAL, default=default=]

The algorithm for the level2 part of the generic pipeline.

Integer apid [INPUT, OPTIONAL, default=No default value]

ApID as available in the observation context to be processed. Aspected value: 1028, 1029, 1030, 1031

ObservationContext obs [IO, OPTIONAL, default=(alternative to htp)]

No default value, null allowed. The Observation Context processed , in input Level 0, in Output Level 1

HifiTimelineProduct htp [IO, OPTIONAL, default=(alternative to obs)]

the HifiTimelineProduct to be processed.

Not yet in use for the default algo applied here cal [IN, MANDATORY, default=no default value]

type palStore [Not yet in use for the default algo applied here, MANDATORY, default=no default value]

String db [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]

the database containing the specified obsid.

String sideband [IN, OPTIONAL, default=default=both]

the side band to be processed: "USB", "LSB" or "both"

MapContext result [OUT, OPTIONAL, default=default=None]

a MapContext containing the result products for the side bands as well as the cubes produced by the mapping task

Boolean gui [IN, OPTIONAL, default=default=False.]

only used for a test progress bar. TBD . If the progress bar is displayed or not.

1.90. MkFluxHotCold

Full Name:	herschel.hifi.pipeline.generic.MkFluxHotCold
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import MkFluxHotCold

Description

The module MkFluxHotCold computes the bandpass and receiver temperature for the

intensity calibration of the spectra.

In a first part, the measurements of the hot and cold load included in the input {@link HifiTimeline-Product htp} are identified and suitable groups are build from which receiver temperature and bandpass are calculated at suitable grid points (grid points along the observation time axis). Note that as a minimum requirement for the task to be successful data from both, hot and cold load should be included in the input timeline product.

For an identification of hot and cold measurements, we adopt the following scheme:

- Datasets with sds_type 'hot': all measurements are treated as hot measurements.
- Datasets with sds_type 'cold': all measurements are treated as cold measurements.
- Datasets with sds_type 'hc': typically includes both hot and cold measurements. Here, by default, we distinguish hot and cold measurements from the chopper position. Chopper values larger than 6.3 indicate a 'hot' - values below a 'cold' measurement. Alternatively, the buffer value (values 1,2) can be used. Set the signature keyword 'indicator' to 'buffer'. A value '1' is treated as 'hot' and, consequently, the value '2' as 'cold'. A further alternative is just to assume a suitable pattern of the form hot-cold-cold-hot in case 'isABBA' is not set or set to true or hot-cold-hot-cold-...ins case 'isABBA' is set to False.

Basically, the average flux for the hot and the cold measurements are computed from these groups and combined with the coupling coefficients eta_h and eta_c according to the formulas:

- For the receiver temperature:

  ((eta_h + y*eta_c - yfactor)*j_h - (eta_h + y*eta_c - 1)*j_c)/(yfactor - 1) where the factor y denotes the ratio of the average hot and the average cold flux data and j_h and j_c the temperature of the thermal radiation field.

- For the bandpass:

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

In addition to creating the spectra with the system temperature, a scalar system temperature is constructed which is defined as the median of a tsys spectrum where all the subbands are concatenated. This scalar system temperature is included in the tsys and the bandpass datasets as an additional column ("tsys_median").

Special attention is given to when different LO frequencies are observed (in the frequency switch mode but also in the spectral surveys). Each LO frequency up to within a given tolerance) leads to a different entry in the resulting calibration product (of type `CalFluxHotCold`).

A calibration product of type GenericPipelineCalProduct which contains the coupling coefficients eta_h and eta_c is needed. Either you can pass such a calibration product to the task by setting the value of the task parameter `coupling` accordingly. Alternatively, you can pass the calibration context

included in the observation context (`obs.calibration`). Once you set the task parameter `calibration` with a calibration context you don't need to set the coupling parameter.

A validation mechanism checks the spectra to be subtracted from each other and, if needed, sets in the result data a row flag. The default mechanism inspects the mixer currents (columns 'MJC_hor' or 'MJC_Ver', respectively) and tests their relative deviation (of the hot and cold scans to be combined) to be less than a given tolerance. Once the tolerance is exceeded, for the result spectrum a row flag (bit 17) is set. The tolerance can be set by specifying the 'validatorTolerance'-parameter. By default it is set to 0.025. Alternatively, you can pass a calibration product (typically obtained from the calibration tree) with band-specific tolerances to be applied ('mixerCurrentTolerances'). Yet another possibility is to pass the whole calibration context - the task retrieves the appropriate product with the tolerances. Furthermore, you can use the parameter 'calibration' for that purpose. Once a product of type `GenericPipelineCalProduct` is obtained, the appropriate tolerance is looked up from a table (associated with the given date) from a column named 'mkFluxHotCold'. This checking mechanism can be overruled by passing as 'validator'-parameter an instance of the {@link DataValidator} interface. When setting this parameter to None, no checking is done.

Examples

Example 1: From jide:

```
calib=mkFluxHotCold(htp=htp)
```

Example 2: From jide:

```
# coupling a product of type GenericPipelineCalProduct with the coupling
efficiencies included
calib=mkFluxHotCold(htp=htp, coupling=coupling)
```

Example 3: From jide:

```
calib=mkFluxHotCold(htp=htp, validatorTolerance=0.1)
```

Example 4: From jide:

```
# tolerance of a product of type GenericPipelineCalProduct with the mixer
current tolerances
calib=mkFluxHotCold(htp=htp, coupling=coupling, validatorTolerance=tolerance)
```

Example 5: From jide:

```
# obs an observation context: both, the coupling efficiencies and the mixer
curerent tolerance are obtained from it.
calib=mkFluxHotCold(htp=htp, coupling=obs.calibration,
validatorTolerance=obs.calibration)
# or, equivalent to the above, but much simpler
calib=mkFluxHotCold(htp=htp, calibration=obs.calibration)
```

Example 6: From jide:

```
calib=mkFluxHotCold(htp=htp, indicator='buffer')
```

API Summary

Properties

HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default value]

Properties
CalFluxHotCold **cal** [OUTPUT, OPTIONAL, default=no default value]
Object **coupling** [INPUT, OPTIONAL, default=no default value]
MapContext **calibration** [INPUT, OPTIONAL, default=no default value]
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]
String **indicator** [INPUT, OPTIONAL, default=no default value]
Boolean **isABBA** [INPUT, OPTIONAL, default=no default value]
Boolean **isFSwitch** [INPUT, OPTIONAL, default=no default value]
Object **validatorTolerance** [INPUT, OPTIONAL, default=0.025]
DataValidator **validator** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default value]

CalFluxHotCold cal [OUTPUT, OPTIONAL, default=no default value]

The calibration output product of the this task with the spectra constituting the bandpass and receiver temperature.

Object coupling [INPUT, OPTIONAL, default=no default value]

Parameter to pass a product with the coupling efficiencies included. Alternatively pass a calibration context.

MapContext calibration [INPUT, OPTIONAL, default=no default value]

Parameter to pass a calibration context from which all the calibration input (coupling efficiencies, mixer current tolerances) can be obtained.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Configuration parameter that can be passed to the product

String indicator [INPUT, OPTIONAL, default=no default value]

Allows to specify the identification scheme for distinction of hot/cold measurements: available values are 'Chopper', 'buffer', 'pattern'.

Boolean isABBA [INPUT, OPTIONAL, default=no default value]

In case you have chosen 'pattern' as the way to identify the Hot/Cold measurements, you here can specify whether it is a pattern of the form hot-cold-cold-hot (isABBA=True) or hot-cold-hot-cold (isABBA=False).

Boolean isFSwitch [INPUT, OPTIONAL, default=no default value]

Indicate whether the observation is a FSwitch so that the data is tested for the correct number of different LOFrequencies.

Object validatorTolerance [INPUT, OPTIONAL, default=0.025]

The tolerance to be used for the default validator that checks the difference in the mixer currents and sets a row flag once the mixer currents found in the science data and the bandpass deviate by more than the tolerance from each other. A relative distance measure is used to quantify the deviation (`2abs(x1-x2)/(abs(x1)+abs(x2))`). Alternatively, a calibration object of type `GenericPipelineCalProduct`can be passed from which the band-specific tolerances will be extracted. The tolerances will be looked up from the table in the column named 'mkFluxHotCold'.

DataValidator validator [INPUT, OPTIONAL, default=no default value]

Custom validator that can be passed to the task to check source and ref that is subtracted from each other and flag data once the tolerance exceeds a given tolerance. Note that the validatorTolerance cannot be used to specify this threshold. Furthermore, the default validator is overwritten once a validator or None is passed here.

History

- ...

1.91. MkFreqGrid

Full Name:	herschel.hifi.pipeline.generic.MkFreqGrid
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import MkFreqGrid

Description

Computes a frequency scale which will be used for resampling.

A frequency scale is determined for each frequency group separately. Within the frequency groups, a common scale is computed for each subband / segment. The following procedure is adopted:

- Assume the frequency vector is called 'f', and the stepsize is 's'. Then the min frequency is determined by looping over all the scans within the frequency group but keeping the subband fixed. Similarly, a maximum frequency is computed.
- These minimum and maximum frequencies (minF and maxF, respectively) are rounded to the nearest integer multiple of the stepsize.
- For frequencies in the LSB, the frequencies are first mirrored to the upper sideband so that the rounding procedure result in equally shaped new frequency grids. For the mirroring, we also check whether a velocity correction has been applied and the LO frequency is adjusted accordingly (see the meta data fields 'isVelocityCorrected' or 'velocity_user').
- From these rounded minimum and maximum frequencies we compute the number of grid points (n) in the output grid.
- 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 size stepsize.

In case no stepsize is specified, the stepsize is set to 0.5 for WBS. For HRS, the stepsize is computed by using the following scheme: From the minF and the maxF determined for the frequency group and the number of channels (n), the stepsize s is set to s = (maxF-minF) / (n-1) s = 1 / max(1,round(1/s))

Example

Example 1: From jide:

```
# spectra: the timeline product provided as input
res = mkFreqGrid(htp=spectra)
res = mkFreqGrid(htp=spec, stepsize=0.5) # this will assume that the wave
scale of the spectra is expressed in MHz.
res = mkFreqGrid(htp=spec, stepsize=0.5, unit="MHz")
res = mkFreqGrid(htp=spec, stepsize=[0.2,0.1,0.3])
# using the defaults specified in the xml-files:
params = PipelineConfiguration.getConfig(spec)
res = mkFreqGrid(htp=spec, params=params)
```

API Summary

Jython Syntax

```
See example below.
```

Properties

HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default value]

Properties
CalFreqGrid **cal** [OUTPUT, MANDATORY, default=no default value]
Object **stepsize** [INPUT, OPTIONAL, default=no default value]
String **unit** [INPUT, OPTIONAL, default=no default value.]
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default value]
The input timeline product.

CalFreqGrid cal [OUTPUT, MANDATORY, default=no default value]
The CalFreqGrid task is a product that contains for each frequency group a TableDataset with the columns providing the frequency grids for the different subbands/segments. Some convenience methods are added to facilitate the access to the grids. The columns inherit the units and the naming of the frequency grids found in the input data.

Object stepsize [INPUT, OPTIONAL, default=no default value]
The step size defined for the target grid. You can specify a single double or an array of doubles, one value for each subband.

String unit [INPUT, OPTIONAL, default=no default value.]
Specify the unit the stepsize is expressed in. If no unit is specified it is assumed that the unit of the stepsize is the same as the unit of the wave scale. Typical values are "GHz" or "MHz".

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]
Configuration parameter that can be passed to the product

History

- `2009-05-08 - meli`: Initial version.

1.92. MkHrsBadChans

Full Name:	herschel.hifi.pipeline.hrs.MkHrsBadChans
Alias:	MkHrsBadChans
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import MkHrsBadChans
Category:	HIFI pipeline task

Description

Step2b: Task which checks Internal Tests of HRS to find the bad channels.

Gets the comparison thresholds values, and compares the internal correlation functions to their theoretical values for internal tests. This Task can not be run on science Datasets.

This Task should be run after every Short Functional Tests engineering campaign, and should populate the calibration database with the CalHrsBadChans product outside the standard running pipeline.

IMPORTANT NOTE: This Task has not been yet implemented.

Remarks: This module will strongly depend on the AOT. For instance only one Internal Test can be done, or maybe the 3 (Sine, SquareS, SquareM) corresponding to the current resolution. If all tests are done, then the errorFlag columns will be errorFlag1[Ntest], errorFlag2[Ntest]. The values into errorFlags can be boolean values, or integer values with internal bits which correspond to the boolean value of each Internal Test. If the errorFlags columns are created, meaning that some channels are in error, then the rawCF of the "science" DS have to be truncated (TBD). The determination of the bad channels could be improved in "correlation" mode, by using a special test (all channels cascaded in one face of HRS, the same in the other face, and comparison between both faces). Then one could be sure that the Correlation mode does not introduce new bad channels compared to Internal Tests.

Examples

Example 1: From jide (old style):

```
<code><br>
from herschel.hifi.pipeline.product import *
from herschel.hifi.pipeline.hrs import *
htp = HifiTimelineProduct(df, hk)
htp = DoHrsSubbands()(htp=htp)
CalHrsBadChans = MkHrsBadChans()(htp=htp, -"threshold"=4)
</code>
```

Example 2: From jide (new style):

```
<code><br>
from herschel.hifi.pipeline.hrs.PipelineModules import *
htp = HifiTimelineProduct(df, hk)
doHrsSubbands(htp=htp)
doHrsBadChans(htp=htp, cal=calHrsBadChans)
calHrsBadChans = mkHrsBadChans(htp=htp, -"threshold"=4)
</code>
```

API Summary

Jython Syntax
`See example below.`

Properties
HifiTimelineProduct **htp** [INOUT, MANDATORY, default=No default value]
Integer **threshold** [INPUT, OPTIONAL, default=2]
CalHrsBadChans **cal** [OUTPUT, MANDATORY, default=No default value]

API details

Properties

HifiTimelineProduct htp [INOUT, MANDATORY, default=No default value]
The HifiTimelineProduct which contains the technical Datasets.

Integer threshold [INPUT, OPTIONAL, default=2]
A modifier for defining the threshold values for the comparison.

CalHrsBadChans cal [OUTPUT, MANDATORY, default=No default value]
The Product which contains the bad channels.

History

- 2005-04-29 - OCJ: creation
- 2006-09-11 - OCJ: HifiVectorDataset replaced by HifiPipelineProduct
- 2006-12-07 - OCJ: HifiPipelineProduct replaced by HifiTimelineProduct

1.93. MkOffSmooth

Full Name:	herschel.hifi.pipeline.generic.MkOffSmooth
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import MkOffSmooth

Description

This module allows to calibrate a baseline by processing the measurements of an OFF position in sky.

The following steps are carried through:

- First, the OFF datasets are identified within the observation: type 'science' and 'isLine' false.
- Second, the OFF datasets are smoothed. At the moment, a Gaussian smoothing filter is used. By default, the width is proportional to the square root of the (# ON scans / # OFF scans).
- Finally, the smoothed measurements included in a OFF dataset are averaged. This means that the original number of spectra included in the an OFF dataset is reduced to one (on a per dataset basis). In this way, for each dataset, associated with a particular phase in the observation, one baseline is obtained. This baseline is included in one of the datasets of the resulting calibration product - teh dataset with the same group id. The time stamp of the single baseline is defined as the average of the observation times of the measurements included in the OFF dataset at the input level.

The baselines included within these datasets are ordered in observation time.

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default value]
CalOffBaseline **cal** [OUTPUT, OPTIONAL, default=no default value]
PipelineConfiguration **params** [INPUT, OPTIONAL, default=no default value]
String **mode** [INPUT, OPTIONAL, default=default value filter]
Boolean **ignore** [INPUT, OPTIONAL, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default value]

The input data.

CalOffBaseline cal [OUTPUT, OPTIONAL, default=no default value]

The calibration product with the baselines - for each LoSetting a different Spectrum2d dataset is provided.

PipelineConfiguration params [INPUT, OPTIONAL, default=no default value]

Configuration parameter that can be passed to the product

String mode [INPUT, OPTIONAL, default=default value filter]
Set the mode by which the baseline should be calculated ('avg' / 'filter' / 'fitter').

Boolean ignore [INPUT, OPTIONAL, default=no default value]
If set to True no baseline is calculated.

History

- `2008-04-28 - meli`: New version.

1.94. MkSidebandGain

Full Name:	herschel.hifi.pipeline.generic.MkSidebandGain
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import MkSidebandGain

Description

Prepares a helper object that provides the means to compute the IF- and LO-frequency dependent sideband gains ratios.

The output is specified as an interface type (only the interface is needed by the {@link DoSidebandGain}-task). This will allow to include different implementations and provide the means to have them easily configured.

Example

Example 1: from herschel.hifi.pipeline.generic import MkSidebandGain

```
mkSidebandGain = MkSidebandGain()
gains = mkSidebandGain(htp=htp, shape=shapes, level=levels)
```

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, MANDATORY, default=no default value]
CalSidebandCoeff **cal** [OUT, MANDATORY, default=no default value]
MapContext **calibration** [INPUT, OPTIONAL, default=no default value]
GenericPipelineContext **level** [INPUT, MANDATORY, default=no default value]
GenericPipelineContext **shape** [INPUT, MANDATORY, default=no default value]

API details

Properties

HifiTimelineProduct htp [INPUT, MANDATORY, default=no default value]

The input {@link HifiTimelineProduct} which is used to look up the detector band and the startDate of the observation. This is information is needed to look up the associated information from the 'level' and 'shape' calibration products passed as input parameters to the task (and typically retrieved from the calibration tree).

CalSidebandCoeff cal [OUT, MANDATORY, default=no default value]

The helper object (which is not a persistable product) that allows to compute the sideband gains ratios. This object is passed as input parameter to the {@link DoSidebandGain}-task.

MapContext calibration [INPUT, OPTIONAL, default=no default value]

Parameter to pass a calibration context from which all the calibration input (shape and level tables for the sideband gains) can be obtained.

GenericPipelineContext level [INPUT, MANDATORY, default=no default value]

The calibration product that contains tables with the LO dependent gains coefficients level. Different tables correspond to different observation times (the observations have been taken at). This allows to account for potential instrument drifts.

GenericPipelineContext shape [INPUT, MANDATORY, default=no default value]

The calibration product that contains tables with the IF dependent shape of the gains coefficients. Different tables correspond to different observation times (the observations have been taken at). This allows to account for potential instrument drifts.

History

- `2009-08-28 - Melchior`: Initial - replaces old implementation that have not been used.

1.95. MkSpur

Full Name:	herschel.hifi.pipeline.wbs.MkSpur
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import MkSpur
Category:	task

Description

A task to identify spurs in WBS HTPs

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

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.

Example

Example 1: Simple example.

```
spurTable = mkWbsSpur(htp)
```

API Summary

Properties
INPUT **htp** [, Mandatory, default=No default value]
INPUT **threshold** [Double, Optional, default=800]
INPUT **sigma** [Double, Optional, default=10.0]
INPUT **fwhm** [Double, Optional, default=15.0]
OUTPUT **result** [, MANDATORY, default=no default value]

API details

Properties

INPUT htp [, Mandatory, default=No default value]

The HTP to process

INPUT threshold [Double, Optional, default=800]

The flux above which the WBS is considered saturated

INPUT sigma [Double, Optional, default=10.0]

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 10.0was determined from TV/TB and Gascell observations.

INPUT fwhm [Double, Optional, default=15.0]

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.

OUTPUT result [, MANDATORY, default=no default value]

The list of spurs

1.96. MkWbsBadPixels

Full Name:	herschel.hifi.pipeline.wbs.MkWbsBadPixels
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import MkWbsBadPixels
Category:	HIFI pipeline task

Description

Step2: Task that check for saturated pixels

in the SpectrumDataset.

A CalWbsBadPixel can be passed as supplementary parameter through the input output "cal".

The CalWbsBadPixel contains: value for the maximum pixel level, the threshold values for saturation, and the BadPixel mask to be applied to the HifiSpectrumDataset.

If a CalWbsBadPixel is not passed a new one is created.

After the calculation, the saturated pixel position are stored in the flag.

Then, if a channel is saturated a number of time greater then the values CalWbsBadPixel.getThresholdRepetition(), it is marked as new bad pixel. The new bad pixels are merged with the initial mask passed with CalWbsBadPixel .

Examples

Example 1: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
hk=AccessPacketTask()(obsid=obsid,apid=1026)
spectra=HifiTimelineProduct(hdf,hk)
spectra = DoWbsScanCount(htp=spectra)       #step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=Bool1d(8192,1)) #step 2
```

Example 2: From jide:

```
from herschel.hifi.pipeline.wbs import *
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra)              #step 1
calPixel=MkWbsBadPixels()(spectra)   #step 2
```

API Summary

Jython Syntax

```
See example below.

<p>
```

Properties

{@link HifiTimelineProduct} **"htp"** [IN, Yes mandatory, default=no default value]

Properties
{@link CalWbsBadPixel} **"cal"** [IO, MANDATORY, default=no default value]

API details

Properties

{@link HifiTimelineProduct} "htp" [IN, Yes mandatory, default=no default value]

No default value.

Provides the HifiTimelineProduct to be anylized.

{@link CalWbsBadPixel} "cal" [IO, MANDATORY, default=no default value]

Provides the initial bad PixelList, the parameters for the saturation thresholds and return a bad pixel in function of the saturated pixels.

History

- 15-May-2005 AL : Javadoc and help completed
- parameter renamed for more compatibility with hrs
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.97. MkWbsFluxAtten

Full Name:	herschel.hifi.pipeline.wbs.MkWbsFluxAtten
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import MkWbsFluxAtten
Category:	HIFI pipeline task

Description

Step10: Attenuator settings analysis.

Provides the technical SpectrumDataSet that will be used to calculated the Intensity calibration in function of the attenuator settings .

Examples

Example 1: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf) spectra =
DoWbsScanCount()(htp=spectra) #step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask) #step 2
spectra = DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1 -) #step 3
dd=DoWbsDark() #step 4
spectra = dd(htp=spectra,darkKind=LongParameter(DoWbsDark.DARK1_2))
spectra = DoWbsNonlin()(htp=spectra,cal=MyTablePolyn) #step 5
mkz=MkWbsZero() #step 6
zeroCal=mkz(htp=spectra,interp=LongParameter(zeroCal.NONE))
zerocheck=mkz.zeroCheck
spectra =DoWbsZero()(htp=spectra,cal=zeroCal) #step 7
mkf=MkWbsFreq() #step 8
freqCal=mkf(htp=spectra,interp="PREVIOUS",COMB_FIRST_LINE_POSITION=160,
COMB_LINES_STEP=175,
     COMB_THRESHOLD=500)
combCheck=mkf.combCheck #equivalent to: combCheck=freqCal.check
DoWbsFreq()(htp=spectra,cal=freqCal) #step 9
att=MkWbsFluxAtten()(htp=spectra,cal=CalWbsAttSpecific()) #step 10
```

Example 2: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra) #step 1
calPixel=MkWbsBadPixels()(spectra) #step 2
MkWbsBadPixels()(spectra,calPixel) #step 3
DoWbsDark()(spectra) #step 4
DoWbsNonlin()(spectra) #step 5
zeroCal=MkWbsZero()(spectra) #step 6
DoWbsZero()(spectra,zeroCal) #step 7
freqCal=MkWbsFreq(spectra) #step 8
DoWbsFreq()(spectra,freqCal) #step 9
att=MkWbsFluxAtten()(spectra) #step 10
```

API Summary

Jython Syntax

```
See example below.
<p>
```

Properties
{@link HifiTimelineProduct} **"htp"** [IN, No mandatory, default=yes]
{@link CalWbsAttSpecific } **"cal"** [OUT, Provides an, default=no default value]

API details

Properties

{@link HifiTimelineProduct} "htp" [IN, No mandatory, default=yes]

default value.

{@link CalWbsAttSpecific } "cal" [OUT, Provides an, default=no default value]

IntensityCalibration that can be applied to the SpectrumDataset

History

- 15-May-2005 AL : Javadoc and help completed parameter renamed for
- more compatibility with hrs 14-Jul-2005 AL : Implementation
- completed
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.98. MkWbsFreq

Full Name:	herschel.hifi.pipeline.wbs.MkWbsFreq
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import MkWbsFreq
Category:	HIFI pipeline task

Description

Step8: Derive frequency scale from comb spectra.

Remarks:If there will be different Column for odd/even pixel, they have to be merged before Comb Analysis. CalibrationObject request field for COMB :

COMB_LINE_WIDTH =3 ; Width of a line in pixel

COMB_LINES_FREQUENCY_STEP = 100 distance between lines in MHz;

COMB_NUMBER_OF_LINES = 11 Number of lines in each CCD ;

COMB_LINES_STEP = 175 Distance between lines in pixels;

COMB_LINES_STEP_TOLLERANCE =15 Distance tollerance between lines in pixels;

COMB_MAX_N_SPIKE_CHECK = 15 Maximum of number of Spikes allowed before to rise a warning flag;

COMB_GAUSSIAN_RANGE = 20 Range in pixel used used to fit the line (-range,+range);

COMB_POLYNOMIAL_DEGREE = 4 Degree of the polynomial used to calculate the Frequency;

COMB_NOISE_RANGE = COMB_LINES_STEP/8;

COMB_START_CCD = 20; begin of CCD when start to search lines

COMB_END_CCD = 2027; end of CCD when start to search lines

COMB_THRESHOLD = 4000; Minimum value allowed to be a line

COMB_CHECK_DR = 3000 Minimum values for Dynamic Range of CCDs

COMB_CHECK_RESOLUTION = 1.1Maximum value for the Resolution of a CCDs in MHz

COMB_CHECK_EFFICIENCY = 35;Minimum value for the efficiency of a CCDs [%]

COMB_CHECK_RIPPLE =10 Maximum value for the ripple of a CCDs [dB]

FrequencyCalibration request field: pixel=polynomial(nu,Coefficient)(t) frequency=polynomial(pixel,Coefficient)(t) General COMB CheckField (Unable to fit) HealthCheck request field: Resolution Values/ Check. Dynamic Range Values/ Check. Efficiency Values/ Check. Ripple [dB]Values/ Check. Gaussians Fit values.[Amplitude,Resolution,Position,Power,Frequency] HifiSpectrumDataset request field: Name/position of spikes flags

Examples

Example 1: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf) spectra =
```

Example 1: From jide: from herschel.hifi.pipeline.wbs import *

```
DoWbsScanCount()(htp=spectra) #step 1
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask) #step 2
spectra = DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1 -) #step 3
dd=DoWbsDark() #step 4
spectra = dd(htp=spectra,darkKind=LongParameter(DoWbsDark.DARK1_2))
spectra = DoWbsNonlin()(htp=spectra,cal=MyTablePolyn) #step 5
mkz=MkWbsZero() #step 6
zeroCal=mkz(htp=spectra,interp=LongParameter(zeroCal.NONE))
zerocheck=mkz.zeroCheck
spectra =DoWbsZero()(htp=spectra,cal=zeroCal) #step 7
mkf=MkWbsFreq() #step 8
freqCal=mkf(htp=spectra,interp="PREVIOUS",COMB_FIRST_LINE_POSITION=160,
COMB_LINES_STEP=175,
 COMB_THRESHOLD=500)
combCheck=mkf.combCheck #equivalent to: combCheck=freqCal.check
```

Example 2: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra) #step 1
calPixel=MkWbsBadPixels()(spectra) #step 2
MkWbsBadPixels()(spectra,calPixel) #step 3
DoWbsDark()(spectra) #step 4
DoWbsNonlin()(spectra) #step 5
zeroCal=MkWbsZero()(spectra) #step 6
DoWbsZero()(spectra,zeroCal) #step 7
freqCal=MkWbsFreq(spectra) #step 8
```

Example 3: Use the tuning to set a different minimum of lines detected:

```
tuning= CalWbsFreqTuning()
freqCal=MkWbsFreq(spectra)
```

API Summary

Jython Syntax
See example below. * <p>

Properties
{@link HifiTimelineProduct} **"htp"** [IN, Yes mandatory, default=No]
{@link String} **"interp"** [IN, No mandatory, default=yes]
{@link Integer} **"COMB_FIRST_LINE_POSITION"** [IN, No mandatory, default=yes default]
{@link Integer} **"COMB_LINES_STEP"** [IN, No mandatory, default=yes default value=]
{@link Double} **"COMB_THRESHOLD"** [IN, No mandatory, default=yes default value=]
{@link CalWbsFreq} **"cal"** [OUT, Provides the, default=no default value]
{@link CalWbsFreqCheck} **"combCheck"** [OUT, No mandatory, default=no default value]
{@link Boolean} **"use_bad"** [IN, No mandatory, default=Default=true]

API details

Properties

{@link HifiTimelineProduct} "htp" [IN, Yes mandatory, default=No]

default value.

Provides the technical SpectrumDataSet with the Comb that will be used to calculated the Frequency calibration.

{@link String} "interp" [IN, No mandatory, default=yes]

default value= "NEAREST" Provides the Parameters used to decide which interpolation scheme will be applied to compute the comb fitting and interpolation scheme

{@link Integer} "COMB_FIRST_LINE_POSITION" [IN, No mandatory, default=yes default]

value= null. If the values is different from null overwrite the tuning in Parameter used as CalWbsFreqTuning. It is used to guess for the first line in COMB. Useful when the number of lines automatically detected are lesser then the expected, to guess hypothetical lines.

{@link Integer} "COMB_LINES_STEP" [IN, No mandatory, default=yes default value=]

null. If the values is different from null overwrite the tuning in Parameter used as CalWbsFreqTuning. It is used to guess for distance in pixels between lines in COMB. Useful when the number of lines automatically detected are lesser then the expected, to guess hypothetical lines.

{@link Double} "COMB_THRESHOLD" [IN, No mandatory, default=yes default value=]

null. If the values is different from null overwrite the tuning in Parameter used as CalWbsFreqTuning. It is used as threshold for line detection in COMB: as minimum value allowed to be a COMB line.

{@link CalWbsFreq} "cal" [OUT, Provides the, default=no default value]

FrequencyCalibration .

{@link CalWbsFreqCheck} "combCheck" [OUT, No mandatory, default=no default value]

Equivalent to: MkWbsFreq.cal.check See also the CalWbsFreqCheck for a detailed description.

{@link Boolean} "use_bad" [IN, No mandatory, default=Default= true]

This parameter decide if the bad Combs should be used in the frequency calibration.

use_bad=false: bad combs (comb where all lines was not found) will not used to compute frequency calibration. If there will be not valid combs a default calibration is set in the output and an error will be raised. use_bad=true: also bad combs (comb where all lines was not found) will used to compute frequency calibration. Provides the general checks and pixels checks on the Combs .

The combCheck contains the result of the method fitCcd

The `fitCcd` fit a ccd to compute the parameter of the Gaussians.

{@link Boolean} "use_bad" [IN, No mandatory, default=Default= true]

param: channel an `Int1d` with the position in the ccd of the lines

param: offsetPixels an `int` the absolute position of the first pixel of the ccd

param: ccd a `Double1d` value

param: offsetFreq a `double` the frequency of the first line

return: a `CompositeDataset` the result of the fit, it contains:

- MetaData: "dynamic range","efficiency","ripple","RMS of (real comb freq - fit)" (`DoubleParameter`)s

- TableDataset "gaussian" with the Columns:"Standard Deviation","Amplitude" ,"Resolution","Position","Power","Frequency".

- TableDataset "polynomials" with the Columns:"C: pixel=polynomial(nu,C)" ,"C: frequency=polynomial(pixel,C)".

- ArrayDataset "comb fitted" The `fitCcd` method use the following algorithm:

For each line a number of pixel equals to 2*DefaultValues.COMB_GAUSSIAN_RANGE +1 are selected, we refer to this selection as LineCut.

Then to each LineCut is subtracted its minimum value.

Then each LineCut is fitted with the `LevenbergMarquardtFitter` with a `GaussModel`.

From each "gaussian" fit are calculated: "Standard Deviation","Amplitude","Resolution","Position" ,"Power","Frequency".

Two "polynomials" are calculated from the "Position" and the "Frequency" of the lines to get the pixels in function of the frequencies and to get the frequencies in function of the pixels. The values of the coefficients are put in the "C: pixel=polynomial(nu,C)" and "C: frequency=polynomial(pixel,C)" Columns of the `TableDataset` "polynomials"

From the difference between the real frequency of the lines and the frequency calculated with the polynomial is calculated the DoubleParameter "RMS of (real comb freq - fit)"

From the ccd the LineCut are removed and than it is calculated the dynamic range and set in the `MetaData` "dynamic range" as `DoubleParameter`

From the "Power" is calculated the power reduced removing the first and last line, then the average is put in the `MetaData` "efficiency" as `DoubleParameter`.

From the the power reduced maximum and minimum is calculated the Ripple [db]: 10 *ln(maxPower/minPower) and put it in `MetaData` "ripple" as `DoubleParameter`.

The gaussian calculated from the fitter are put in the "comb fitted" `ArrayDataset`.

{@link Boolean} "use_bad" [IN, No mandatory, default=Default= true]
CalWbsFreq

History

- 15-May-2005 AL : Javadoc and help completed parameter renamed for
- more compatibility with hrs 14-Jul-2005 AL : First implementation
- completed.
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.99. MkWbsZero

Full Name:	herschel.hifi.pipeline.wbs.MkWbsZero
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs import MkWbsZero
Category:	HIFI pipeline task

Description

Step6: Check the zeros and compute an interpolation function to

represent zeros for each time.

ScanInterpolator.PREVIOUS = "PREVIOUS"; //AL 12-11-2006 New ScanInterpolator.NEAREST = "NEAREST"; ScanInterpolator.LINEAR = "LINEAR"; ScanInterpolator.CUBIC_SPLINE = "CUBIC_SPLINE"; //AL 12-11-2006 New request field :

Double1d thresholds= CalWbsZeroCheck.getThreshold(); double zeroVarianceMaximum = thresholds.get(0); double zeroAverageMaximum = thresholds.get(1); double zeroAverageMinimum = thresholds.get(2); double zeroMaximum = thresholds.get(3); double zeroMinimum = thresholds.get(4); HifiSpectrumDataset request field: Name/position of bad pixels from zero flags

Examples

Example 1: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
spectra =DoWbsScanCount()(htp=spectra)
calPixel=MkWbsBadPixels()(htp=spectra,badPixel=MyBadPixelMask) #step 2
spectra = DoWbsBadPixels()(htp=spectra,cal=calPixel,apply=1 -) #step 3
dd=DoWbsDark() #step 4
spectra =dd(htp=spectra,darkKind=DoWbsDark.DARK1_2)
spectra = DoWbsNonlin()(htp=spectra,cal=MyTablePolyn) #step 5
mkz=MkWbsZero() #step 6
zeroCal=mkz(htp=spectra,interp="NEAREST")
zerocheck=mkz.zeroCheck   (equivalent to zerocheck=zeroCal.check)
```

Example 2: From jide: from herschel.hifi.pipeline.wbs import *

```
hdf=AccessDataFrameTask()(obsid=obsid,apid=1030)
spectra=WbsSpectrumDataset(hdf)
DoWbsScanCount()(spectra) #step 1
calPixel=MkWbsBadPixels()(spectra) #step 2
MkWbsBadPixels()(spectra,calPixel) #step 3
DoWbsDark()(spectra) #step 4
DoWbsNonlin()(spectra) #step 5
zeroCal=MkWbsZero()(spectra) #step 6
```

API Summary

Jython Syntax

```
See example below.
<p>
```

Properties
{@link HifiTimelineProduct}[] **"htp"** [IN, Yes mandatory, default=No]
{@link String} **"interp"** [IN, No mandatory, default=yes]
{@link CalWbsZero} **"cal"** [OUT, Provides the, default=no default value]
{@link CalWbsZeroCheck} **"zeroCheck"** [OUT, No mandatory, default=no default value]

API details

Properties

{@link HifiTimelineProduct}[] "htp" [IN, Yes mandatory, default=No]

default value.

Provides the technical SpectrumDataSet that will be used to calculated the zeros calibration.

{@link String} "interp" [IN, No mandatory, default=yes]

default value = "NEAREST" Provides the Parameters used to decide which interpolation scheme will be applied to compute the zero Calibration

PREVIOUS; No interpolation. Use the most recent Zero (Time of zero < Time Spectrum). If there aren't Zeros with time before, it will use the first zero taken after.

NEAREST; No interpolation. Use the Zero with the closest time to the time of the spectrum.

{@link CalWbsZero} "cal" [OUT, Provides the, default=no default value]

ZeroCalibration .

{@link CalWbsZeroCheck} "zeroCheck" [OUT, No mandatory, default=no default value]

Provides the general checks and pixels checks on the zeros .

It is equivalent to CalWbsZero.getCheck

History

- 15-May-2005 AL : Javadoc and help completed parameter renamed for
- more compatibility with hrs 14-Jul-2005 AL : First implementation
- completed.
- 07-Dic-2006 AL: converted to HifiTimelineProduct and Calibration Product.
- 21-Dic-2006 AL: Uniformed api to generic branch. Updated documentation

1.100. PipelineConfiguration

Full Name:	herschel.hifi.pipeline.generic.PipelineConfiguration
Type:	Java Class -
Import:	from herschel.hifi.pipeline.generic import PipelineConfiguration

Description

Instances of this class provide configuration parameters for the pipeline modules.

The parameters are specified in a xml-configuration file that contains observing mode-specific settings. the file is located on the class path - the default settings are included in the pipeline.generic package, user specific settings can be localized by setting the property 'hifi.pipeline.generic.config'. Note that the file with the user-specific settings should be self-contained - all the default settings are overwritten in this way.

1.101. PipelineTask

Full Name:	herschel.hifi.pipeline.PipelineTask
Alias:	PipelineTask
Type:	Jython Task -
Import:	from herschel.hifi.pipeline import PipelineTask

Description

Task to wrap pipeline algorithms

Example

Example 1: PipelineTask

```
<pre>
from herschel.hifi.pipeline.PipelineTask import *
pipelineTask=PipelineTask()
#
# Example 1, sgenerate an observation context import scratch
# the default db = retrieved import the property: var.database.devel, check
with propgen.
#
obs = pipelineTask(obsid=268435583, db="ds1@iccdb.sron.rug.nl 0 READ")
#
# Note: you can use the dataset inspector to inspect the outcoming
observation context
#
#
# Example 2, like ex. 1, now use of obsMode parameter,
# for ILT data you have to provide the obsMode name because the data itself
does not have it!#
#
obs = pipelineTask(obsid=268516902, db="ilt_fm_5_prop@iccdb1.sron.rug.nl 0
READ", obsMode="HifiPointModeLoadChop",gui=1)
#
#
# Example 3, like ex. 1,  now use of apids parameter,
# pass an apid for level 1 processing (level 0 always process all data -/ 4
apids, is not yet optional)
#
obs = pipelineTask(obsid=268516902, apids=["1030"],
obsMode="HifiPointModeLoadChop")
#
#
# Example 4, like ex. 1 now using your own algorithm for wbs -/hrs or generic:
# one can look -/ edit -/ use the algo"s by loading the following scripts
into jide:
# The WBS pipeline algo is found in the {build_root}/lib/herschel/hifi/
pipeline/wbs/WbsPipelineAlgo.py
# The HRS pipeline algo is found in the {build_root}/lib/herschel/hifi/
pipeline/hrs/HrsPipelineAlgo.py
# The Generic pipeline algois found in the {build_root}/lib/herschel/hifi/
pipeline/generic/GenericPipelineAlgo.py
#
obs = pipelineTask(obsid=268435583, db="ds1@iccdb.sron.rug.nl 0 READ",
wbsAlgo=myWbsAlgo, hrsAlgo=myHrsAlgo,level1Algo=mylevel1Algo)
#
#
#
# Example 6, reprocess an existing observation context
# only reprocess hrs/wbs/generic pipelines:
pipelineTask(obs=obs)
#
#
# Example 7: like ex. 1-6, now only generate level0
```

Example 1: PipelineTask

```
# Note that for level o processing all available apids are processed, i.e. it
# does not react on the apids parameter!
#
obs = pipelineTask(obsid=268516902,  db="ilt_fm_5_prop@iccdb1.sron.rug.nl 0
READ",obsMode="HifiPointModeLoadChop", upToLevel=0)
#
#similar:
#
obs = pipelineTask(obsid=268516902, db="ilt_fm_5_prop@iccdb1.sron.rug.nl 0
READ", obsMode="HifiPointModeLoadChop", upToLevel=1)
#
#
# Example 8: like ex. 1-6, now reprocess all levels incl. level 0
# Note that all calibration and other products (auxilary etc) are not replaced.
#
pipelineTask(obs=obs, fromLevel=-1)
#
#
#
# Example 9: like 1 now providing a tm_version_map
# in case the selecated database requires a different mission phase than
defined in your
# binstruct property: hcss.binstruct.mib.pal.tm_version_map (hifi default
= -"ilt-fm")
#
obs = pipelineTask(obsid=268435583, db="ilt_par_7_prop@iccdb.sron.rug.nl 0
READ", tmVersion="ilt-par")
#
#
# Example 10:
# clear CachedStoreHandler to avoid a block by this Handler due to none
closed stores
# note: closes ALL stores availbale in this cache and thus might affect other
applications
# running in this session.
#
obs = pipelineTask(obsid=268435583, db="ds1@iccdb.sron.rug.nl 0 READ",
upToLevel=0.5, clearCachedStoreHandler=1)
#
#
# Example 11: like 1 now providing your own palStore
# Note that the pipeline tasks needs the hifi-cal@iccdb pool (or simple or
lstore copy) to generate the level0 product
# so one should always include it:
#
myStore = herschel.ia.pal.ProductStorage()
myStore.register(SimplePool.getInstance("my-dedicated-pool-to-be-applied")
#
# the next are registered to retrieve your calibration data:
# local:
myStore.register(SimplePool.getInstance("hifi-cal")) # or hifi-cal_dev for ex.
# or, server pool:
import herschel.share.util.Configuration
Configuration.setProperty("hifical", self.cal)
myStore.register(DbPool.getInstance("hifical"))
#
obs = pipelineTask(obsid=268435583, db="ds1@iccdb.sron.rug.nl 0 READ",
palStore = myStore)
#
#
# to browse the products in the pool:
def startProductBrowser(storage):
    from herschel.ia.pal.browser import ProductBrowser
    result = ProductBrowser.browseProduct(storage)
    return result
#
res = startProductBrowser(myStore)
</pre>
```

API Summary

Properties
array(String) **apids** [IN, OPTIONAL, default=default=all apids]
ObservationContext **obsOut** [OUT, OPTIONAL, default=No default value]
ObservationContext **obs** [IO, OPTIONAL, default=(alternative to obsid)]
Long **obsid** [IN, OPTIONAL, default=No default value]
String **db** [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]
String **cal** [IN, OPTIONAL, default=default= Configuration.getProperty("hcss.binstruct.mib.pal.database")]
String **obsMode** [IN, OPTIONAL, default=default=self determined]
ProductStorage **palStore** [IN, OPTIONAL, default=default= ProductStorage:"hifi-pipeline"]
PyFunction **wbsAlgo** [INPUT, OPTIONAL, default=default=]
PyFunction **hrsAlgo** [INPUT, OPTIONAL, default=default=]
PyFunction **genericAlgo** [INPUT, OPTIONAL, default=default=]
PyFunction **level1Algo** [INPUT, OPTIONAL, default=default=]
PyFunction **level2Algo** [INPUT, OPTIONAL, default=default=]
Boolean **reprocessAllLevels** [IN, OPTIONAL, default=default=False.]
Boolean **gui** [IN, OPTIONAL, default=default=False.]

API details

Properties

array(String) apids [IN, OPTIONAL, default=default=all apids]

the apid that have to be processed

ObservationContext obsOut [OUT, OPTIONAL, default=No default value]

The Observation Context processed until level 2 when the input is just: (obsid).

ObservationContext obs [IO, OPTIONAL, default=(alternative to obsid)]

No default value, null allowed. The Observation Context processed , in is none, level0 or level0_5, out is level1 and level2

Long obsid [IN, OPTIONAL, default=No default value]

Alternative input for the observation to be processed. ObservationContext created on the fly

String db [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]

the database containing the specified obsid.

String cal [IN, OPTIONAL, default=default= Configuration.getProperty("hcss.binstruct.mib.pal.database")]

the remote pal database containing MIB and the calibration products.

String obsMode [IN, OPTIONAL, default=default=self determined]

the observation mode to be used in the pipline. Overwrite the obsMode present in the ObservationContext

ProductStorage palStore [IN, OPTIONAL, default=default= ProductStorage:"hifi-pipeline"]

The store where the result will be stored and where the calibration product can be retrieved.

PyFunction wbsAlgo [INPUT, OPTIONAL, default=default=]

The algorithm for the wbs pipeline.

PyFunction hrsAlgo [INPUT, OPTIONAL, default=default=]

The algorithm for the hrs pipeline.

PyFunction genericAlgo [INPUT, OPTIONAL, default=default=]

The algorithm for the generic pipeline.

PyFunction level1Algo [INPUT, OPTIONAL, default=default=]

The algorithm for the level1 pipeline.

PyFunction level2Algo [INPUT, OPTIONAL, default=default=]

The algorithm for the level2 pipeline.

Boolean reprocessAllLevels [IN, OPTIONAL, default=default=False.]

In case you pass an (adopted) observation context and you want to reprocess level 0 and following switch this one to TRUE

Boolean gui [IN, OPTIONAL, default=default=False.]

If the progress bar is displayed or not.

History

- 12-Feb-08 PZ Add JTAGs for this task
- 07-Mar-08 PZ Add update example for this task
- 28-March-2008 PZ make pipelineTask with API for CS

1.102. QualAssessHifiGenericTask

Full Name:	herschel.hifi.pipeline.generic.QualAssessHifiGenericTask
Alias:	QualAssessHifiGenericTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline.generic import QualAssessHifiGenericTask
Category:	task

Description

Task for quality assessment of level 1 Products

API Summary

Properties
Product **input** [Input, Mandatory, default=No default value]
HifiQualityProduct **output** [Input/Output, Mandatory, default=No default value]

API details

Properties

Product input [Input, Mandatory, default=No default value]

the input Product that contains the metadata information from the pipeline

HifiQualityProduct output [Input/Output, Mandatory, default=No default value]

the predefined HifiQualityProduct in which the quality control data will be stored

History

- 07 Sep 07: Created the task from original QualAssessHifi.java.
- 22 Oct 07: Added more metadata parameters.
- 13 Nov 07: Added more metadata parameters.
- 15 Nov 07: Made child of QualAssessHifiTask; removed redundant lines;
- incorporated javadoc formatting into comments.
- 12 May 08: Adapted to use HifiQualityEnum
- 05 June 2008 copied by HIFI for checking re-use for quality plugin

1.103. QualAssessHifiHrsTask

Full Name:	herschel.hifi.pipeline.hrs.QualAssessHifiHrsTask
Alias:	QualAssessHifiHrsTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline.hrs import QualAssessHifiHrsTask
Category:	task

Description

Task for quality assessment of level 1 Products.

This Task will check for quality control metrics in an input level 1 Product, adds the values to a predefined output HifiQualityProduct, and sets boolean flags to indicate whether the metrics fall within nominal ranges.

History

- `2007-09-07 - Created`: the task from original QualAssessHifi.java.
- `2007-10-22 - Added`: more Metadata parameters.
- `2007-11-13 - Added`: more Metadata parameters.
- `2007-11-15 - Made`: child of QualAssessHifiTask; removed redundant lines
- `2008-05-12 - Adapted`: to use HifiQualityEnum
- `2008-06-05 - copied`: by HIFI for checking re-use for quality plugin

1.104. QualAssessHifiProductTask

Full Name:	herschel.hifi.pipeline.level0.QualAssessHifiLevel0Task
Alias:	QualAssessHifiProductTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline.level0 import QualAssessHifiLevel0Task
Category:	task

Description

Task for quality assessment of level 0 Products

API Summary

Properties
`Product` **`input`** `[Input, Mandatory, default=No default value]`
`HifiQualityProduct` **`output`** `[Input/Output, Mandatory, default=No default value]`

API details

Properties

`Product input [Input, Mandatory, default=No default value]`

the input Product that contains the metadata information from the pipeline

`HifiQualityProduct output [Input/Output, Mandatory, default=No default value]`

the predefined HifiQualityProduct in which the quality control data will be stored

History

- 09-03-2009 Created

1.105. QualAssessHifiTask

Full Name:	herschel.hifi.pipeline.util.QualAssessHifiTask
Alias:	QualAssessHifiTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline.util import QualAssessHifiTask
Category:	task

Description

Parent Task class for other quality assessment Tasks

API Summary

Properties
`Product` **`input`** `[Input, Mandatory, default=No default value]`
`HifiQualityProduct` **`output`** `[Input/Output, Mandatory, default=No default value]`

API details

Properties

`Product input [Input, Mandatory, default=No default value]`

the input Product that contains the metadata information from the pipeline

`HifiQualityProduct output [Input/Output, Mandatory, default=No default value]`

the predefined HifiQualityProduct in which the quality control data will be stored

History

- 15 Nov 07: Created the task from QualAssessSpireLevel05Task.java.
- 12 May 08: Adapted to use SpireQualityEnum
- 05 June copied from HIFI using it for testing quality setup

1.106. QualAssessHifiWbsTask

Full Name:	herschel.hifi.pipeline.wbs.quality.QualAssessHifiWbsTask
Alias:	QualAssessHifiWbsTask
Type:	Java Task -
Import:	from herschel.hifi.pipeline.wbs.quality import QualAssessHifiWbsTask
Category:	task

Description

Task for quality assessment of level 1 Products

API Summary

Properties
Product **input** [Input, Mandatory, default=No default value]
HifiQualityProduct **output** [Input/Output, Mandatory, default=No default value]

API details

Properties

Product input [Input, Mandatory, default=No default value]

the input Product that contains the metadata information from the pipeline

HifiQualityProduct output [Input/Output, Mandatory, default=No default value]

the predefined HifiQualityProduct in which the quality control data will be stored

History

- 07 Sep 07: Created the task from original QualAssessHifi.java.
- 22 Oct 07: Added more metadata parameters.
- 13 Nov 07: Added more metadata parameters.
- 15 Nov 07: Made child of QualAssessHifiTask; removed redundant lines;
- incorporated javadoc formatting into comments.
- 12 May 08: Adapted to use HifiQualityEnum
- 05 June 2008 copied by HIFI for checking re-use for quality plugin

1.107. QualityPlugin

Full Name:	herschel.hifi.pipeline.spg.plugin.QualityPlugin
Type:	Java Class -
Import:	from herschel.hifi.pipeline.spg.plugin import QualityPlugin

History

- 20 Apr 2007: First version.
- 25 Apr 2007: don't run quality on level0 products
- 15 Oct 2007: updated for latest QC tasks
- 18 Mar 2008: SPR-0639; changes to ObservationContext API
- 04-Jun-2008: PAZ first edition for HIFI

1.108. RemoveFlaggedPixels

Full Name:	herschel.hifi.pipeline.util.tools.RemoveFlaggedPixels
Type:	Java Task -
Import:	from herschel.hifi.pipeline.util.tools import RemoveFlaggedPixels

Description

Task to handle flagged pixels by either setting the flux value to NaN, replacing it by a value interpolated from non-flagged neighboring values, or by removing the pixel from the segment.

API Summary

Jython Syntax

```
from herschel.hifi.pipeline.product import HifiMask
removePixel = RemoveFlaggedPixels()
removePixel(htp=htp, HifiMask.BAD_PIXEL)
removePixel(htp=htp, [HifiMask.BAD_PIXEL,HifiMask.SATURATED]) #
Removes the pixels that have both bits at the same time
```

Properties

- HifiTimelineProduct **htp** [INOUT, OPTIONAL, default=no default value.]
- Object **mask** [INPUT, OPTIONAL, default=no default value.]
- String **mode** [INPUT, OPTIONAL, default="NaN".]
- String **interpolator** [INPUT, OPTIONAL, default="linear".]

API details

Properties

HifiTimelineProduct htp [INOUT, OPTIONAL, default=no default value.]

The timeline product to be processed.

Object mask [INPUT, OPTIONAL, default=no default value.]

The flag to handle by this task. It can be a HifiMask or an array of such. You can also specify the string 'any' in which case any flagged pixel (flag value > 0) will be removed.

String mode [INPUT, OPTIONAL, default="NaN".]

Specify how to handle the flagged flux values: 'NaN', 'Interpolate', 'Remove'

String interpolator [INPUT, OPTIONAL, default="linear".]

Specify how to interpolate the flux and weight values (if applicable) when the mode='Interpolate' has been chosen.

History

- 06-July-2009 initial.

1.109. SelectSpectrum

Full Name:	herschel.hifi.dp.dataset.spectrum.SelectSpectrum
Type:	Java Task -
Import:	from herschel.hifi.dp.dataset.spectrum import SelectSpectrum

Description

Task for selecting datasets and/or rows of a timeline product ({@link HifiTimelineProduct} or a dataset ({@link Spectrum2d}).

A timeline product or a dataset can be specififed as input but not both - otherwise a runtime exception is thrown. It is an extension of the SelectSpectrum-task in `herschel.ia.toolbox.spectrum` from which it inherits the selection functionality from within the datasets. An additional selection scheme is available to select datasets from within the `HifiTimelineProduct` by looking at meta data.

In summary, the following selection models are available:

- Selection of the datasets from the timeline product by referring to meta data: use 'selection_meta'.
- Selection of the rows from the (possibly in the previous steps pre-selected) datasets by looking up suitable values in specific (1d-)columns: use 'selection_lookup'.
- Select rows by specifying a list of row numbers: use 'selection_index'. Note that this seems not to be suited when processing timeline products.
- Selection of the rows from a general selection model (such as IntervalSelectionModel): use 'selection'.

Remarks: Note that only either a timeline product or a dataset should be passed as input data. In case a timeline product and a dataset are passed a runtime exception is thrown.

Example

Example 1: from jide

```
from herschel.hifi.dp.dataset.spectrum import SelectSpectrum
select = SelectSpectrum()
selectSpectra = select(htp=product, selection_meta={"sds_type":["science"]})
selectSpectra = select(htp=product, selection_lookup={"bbtype":[6031]})
selectSpectra = select(ds=spectrum, selection_meta={"sds_type":["science"])
```

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, OPTIONAL, default=No default value.]
Spectrum2d **ds** [INPUT, OPTIONAL, default=No default value.]
PyDictionary **selection_meta** [IN, OPTIONAL, default=No default value.]
ISelectionModel **selection** [IN, OPTIONAL, default=No default value.]
PyDictionary **selection_lookup** [IN, OPTIONAL, default=No default value.]

Properties
Boolean **return_single_ds** [INPUT, OPTIONAL, default=False.]
Spectrum2d|HifiTimelineProduct **result** [OUTPUT, OPTIONAL, default=No default value]

API details

Properties

HifiTimelineProduct htp [INPUT, OPTIONAL, default=No default value.]

The timeline product to be selected from.

Spectrum2d ds [INPUT, OPTIONAL, default=No default value.]

The dataset to be selected from.

PyDictionary selection_meta [IN, OPTIONAL, default=No default value.]

Dictionary with the meta data parameters to lookup in the datasets.

ISelectionModel selection [IN, OPTIONAL, default=No default value.]

General selection model to specify the selection criteria.

PyDictionary selection_lookup [IN, OPTIONAL, default=No default value.]

Dictionary with the discrete value criteria by referring to column data within the datasets.

Boolean return_single_ds [INPUT, OPTIONAL, default=False.]

Flag indicating whether a single dataset or a hifi timeline product should be returned.

Spectrum2d|HifiTimelineProduct result [OUTPUT, OPTIONAL, default=No default value]

Result object containing the selected spectra. If the flag 'return_single_ds' is set to true, a timeline product is returned with the datasets which that provide a non-empty selection. Otherwise, a dataset with all the selected spectra is returned.

History

- 01-June-2007 initial.
- 13-July-2007 Javadoc updated.

1.110. SelectSpectrum

Full Name:	herschel.hifi.pipeline.util.tools.SelectSpectrum
Type:	Java Task -
Import:	from herschel.hifi.pipeline.util.tools import SelectSpectrum

Description

Task for selecting datasets and/or rows of a timeline product ({@link HifiProduct} or a dataset ({@link Spectrum2d}).

A timeline product or a dataset can be specififed as input but not both - otherwise a runtime exception is thrown. It is an extension of the SelectSpectrum-task in `herschel.ia.toolbox.spectrum` from which it inherits the selection functionality from within the datasets. An additional selection scheme is available to select datasets from within the `HifiTimelineProduct` by looking at meta data.

In summary, the following selection models are available:

- Selection of the datasets from the timeline product by referring to meta data: use 'selection_meta'.
- Selection of the rows from the (possibly in the previous steps pre-selected) datasets by looking up suitable values in specific (1d-)columns: use 'selection_lookup'.
- Select rows by specifying a list of row numbers: use 'selection_index'. Note that this seems not to be suited when processing timeline products.
- Selection of the rows from a general selection model (such as IntervalSelectionModel): use 'selection'.

Remarks: Note that only either a timeline product or a dataset should be passed as input data. In case a timeline product and a dataset are passed a runtime exception is thrown.

Example

Example 1: from jide

```
from herschel.hifi.pipeline.util.tools import SelectSpectrum
selectHifi = SelectSpectrum()
selectSpectra = selectHifi(htp=product, selection_meta={"sds_type":
["science,hc"]})
dsAllScience = selectHifi(htp=product, selection_meta={"sds_type":
["science"], return_single_ds=True})
selectSpectra = selectHifi(htp=product, selection_lookup={"bbtype":[6031]})
dsAllScienceOn = selectHifi(htp=product, selection_lookup={"bbtype":[6031]},
return_single_ds=True})
```

API Summary

Properties
HifiTimelineProduct **htp** [INPUT, OPTIONAL, default=No default value.]
Spectrum2d **ds** [INPUT, OPTIONAL, default=No default value.]
PyDictionary **selection_meta** [IN, OPTIONAL, default=No default value.]

Properties
Object **selection** [INPUT, OPTIONAL, default=None.]
Boolean **return_single_ds** [INPUT, OPTIONAL, default=False.]
PyList **add_from_metadata** [INPUT, OPTIONAL, default=no default value.]
Spectrum2d|HifiTimelineProduct **result** [OUTPUT, OPTIONAL, default=No default value]

API details

Properties

HifiTimelineProduct htp [INPUT, OPTIONAL, default=No default value.]

The timeline product to be selected from.

Spectrum2d ds [INPUT, OPTIONAL, default=No default value.]

The dataset to be selected from.

PyDictionary selection_meta [IN, OPTIONAL, default=No default value.]

Dictionary with the meta data parameters to lookup in the datasets.

Object selection [INPUT, OPTIONAL, default=None.]

Specification of what point spectra select. Different ways to specify these selections are possible:

- Specify a list of indices (in jython) of the point spectra to select.
- Use a dictionary (in jython) to specify a selection model (lookup for attributes, filter attributes to be included in ranges or intervals).
- Pass a string representation of a jython list or a jython dictionary. The string will be parsed and the prcessing delegated to the two cases above.
- Pass any java instance that implements the SelectionModel interface (for the advanced user).

Boolean return_single_ds [INPUT, OPTIONAL, default=False.]

Flag indicating whether a single dataset or a hifi timeline product should be returned. In case a single dataset should be returned the situation may occur taht not all datasets that should be selected have exactly the same subband lengths. In this situation, the subbands are cropped (from above) to the minimum subband length of the spectra to be selected.

PyList add_from_metadata [INPUT, OPTIONAL, default=no default value.]

If you set `return_single_ds` to True you have the possibility to fill meta data from the different datasets into columns of the resulting dataset. Just specify the list of meta data keys to be added. Currently, only meta data values of type `int, long, double, float, string` and `boolean` are supported. In case the specified meta data key is not in all datasets present you can a default value is entered. These defaults are set to

PyList add_from_metadata [INPUT, OPTIONAL, default=no default value.]

- `long` : maximum long number
- `int` : maximum int number
- `double` : NaN
- `float` : NaN
- `boolean` : false
- `string` : empty string

Note that in particular for boolean parameters caution is needed.

Spectrum2d|HifiTimelineProduct result [OUTPUT, OPTIONAL, default=No default value]

Result object containing the selected spectra. If the flag 'return_single_ds' is set to true, a timeline product is returned with the datasets which that provide a non-empty selection. Otherwise, a dataset with all the selected spectra is returned.

History

- 01-June-2007 initial.
- 13-July-2007 Javadoc updated.
- 15-Mar-2009 Bug fixes, works for HifiProduct

1.111. ViewIVcurveScansTask

Full Name:	herschel.hifi.fpu.ivscan.task.ViewIVcurveScansTask
Alias:	ViewIVcurveScansTask
Type:	Java Task -
Import:	from herschel.hifi.fpu.ivscan.task import ViewIVcurveScansTask
Category:	HIFI Diagnostic Housekeeping

Description

Task for plotting IV-curve scan data from array of TM-packets

This utility allows to view in a controlled way the mixer IV-curves extracted from telemetry packets containing FPU IV parameter scan data.

Invocation starts with a pop-up window showing buttons for stepping manually through the available curves either backward or forward. Additional buttons are available for automatic stepping forward through all available curves or to jump back to the first one. The rate of the stepping is controlled by the 'interval' argument. As most scans are composed of several packets there is a toggle-button to select plotting granularity to either complete curves or to TM packets. The plots may contain (colored) layers for showing preceeding curves. Buttons are available to increase or decrease the number level of plotlayer depth.

The plot window consists of three areas. One with textual information about the latest processed TM-packet, like BbId, ObsId, Time, total plotted points, etc. The other two areas contain each an IV-plot for the Horizontal and the Vertical polarity mixer component respectively.

To quit the plotting and stay in jide: close either the plotting window or the button panel.

Examples

Example 1: Shortest (jide) call with all defaults

```
<pre>
from herschel.hifi.generic.task import *
packets = AccessPacketTask()(obsid=743, apid=1026, sids=[257,266])
from herschel.hifi.fpu.all import *
ViewIVcurveScansTask()(packets)
</pre>
```

Example 2: Full (jide) call with all arguments specified

```
<pre>
from herschel.access import *
Access.setDatabase("ilt_fm_5@iccdb.sron.rug.nl 0 READ")
pk=AccessPacketTask(obsid=268516216, sid=266)
from herschel.hifi.fpu.ivscan.task import ViewIVcurveScansTask
ViewIVcurveScansTask()(input=packets, interval=3000, recall=2, accumulate=1)
</pre>
```

API Summary

Jython Syntax
`ViewIVcurveScansTask(tm_packets)`

Properties
TmSourcePacket[] **input** [INPUT, YES/required, default=No default]
Integer **recall** [INPUT, NO/optional, default=2]
Integer **interval** [INPUT, NO/optional, default=4000]
Boolean **accumulate** [INPUT, NO/optional, default=1]

Limitations

Data has to be either constructed or fetched from a database and sorted prior to calling.

Miscellaneous

This task is derived from ProcessViewIvDiagPacket.java in association with IVscanViewAppl.java (package herschel.hifi.fpu.ivscan) which is to be used in DataFlow environment for real-time (router) purposes.

API details

Properties

TmSourcePacket[] input [INPUT, YES/required, default=No default]

Array of TM-source packets. Only packets containing IV scan reports will be processed, all others will be ignored.

Integer recall [INPUT, NO/optional, default=2]

Recall depth code gives the number of previous plots that will be shown together with the current plot. (Maximum = 7) This value can be adjusted at any time during the viewing.

Integer interval [INPUT, NO/optional, default=4000]

Interval period sets the minimal time in milli-seconds between consecutive plot updates. This will only be used when the run button function is activated.

Boolean accumulate [INPUT, NO/optional, default=1]

If true (=1) the partial curves contained in the TM-packets will be accumulated and assembled into complete IV-curves. Otherwise only the (partial) curves are shown on a per packet basis.

See also

- "HIFI TM PACKET STRUCTURE ICD"

History

- First revision: 2005/10/10 [GJW].
- Uses new ia.gui.plot classes since Rev 1.14 .
- Current $Revision: 1.30 $ of $Date: 2010/04/14 08:13:49 $

1.112. WbsCheckFt

Full Name:	herschel.hifi.wbs.data.scripts.wbs.WbsCheckFt
Alias:	WbsCheckFt
Type:	Jython Task -
Import:	from herschel.hifi.wbs.data.scripts.wbs import WbsCheckFt
Category:	task

Description

Housekeeping plotter

Interactive task to analyze and plot WBS Functinal test

API Summary

Jython Syntax
`WbsCheckFt()(obsid = obsid, apid = apid)  `
`WbsCheckFt()(obsid, apid)`
`WbsCheckFt()()`

Properties
`Long obsid [INPUT, OPTIONAL, default=0]`
`Long apid [INPUT, OPTIONAL, default=0]`
`PlotXY p1 [OUTPUT, OPTIONAL, default=0]`
`PlotXY p2 [OUTPUT, OPTIONAL, default=0]`

API details

Properties

Long obsid [INPUT, OPTIONAL, default=0]
ObsID of data, negative values trigger an interactive selection

Long apid [INPUT, OPTIONAL, default=0]
ApID of HK data, negative values trigger an interactive selection

PlotXY p1 [OUTPUT, OPTIONAL, default=0]

PlotXY p2 [OUTPUT, OPTIONAL, default=0]

History

- `2005-08-27 - Initial`: version generated from Task description parameters
- `2006-06-16 - Changed`: to use new PlotXY

1.113. wbsPipelineTask

Full Name:	herschel.hifi.pipeline.wbs.WbsPipelineTask
Alias:	wbsPipelineTask
Type:	Jython Task -
Import:	from herschel.hifi.pipeline.wbs import WbsPipelineTask
Category:	task

Description

Process

Example

Example 1: #

```
# Examples:
from herschel.hifi.pipeline.wbs.WbsPipelineTask import *
wbsPipelineTask(obs=obs, palStore=palStore)
# or process only one apid:
wbsPipelineTask(apid=1030, obs=obs, palStore=palStore)
# or process an htp:
wbsPipelineTask(htp=htp)
# or load df and hk
htp = wbsPipelineTask(obsid=268516902)
#
```

API Summary

Properties
PyFunction **algo** [INPUT, OPTIONAL, default=default=]
Integer **apid** [INPUT, OPTIONAL, default=No default value]
Long **obsid** [IN, OPTIONAL, default=No default value]
Boolean **step** [INPUT, OPTIONAL, default=No default value]
ObservationContext **obs** [IO, OPTIONAL, default=(alternative to htp)]
HifiTimelineProduct **htp** [IO, OPTIONAL, default=(alternative to obs)]
Not Used **cal** [IN, MANDATORY, default=no default value]
type **palStore** [NOT Used, MANDATORY, default=no default value]
String **db** [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]
Boolean **gui** [IN, OPTIONAL, default=default=False.]

API details

Properties

PyFunction algo [INPUT, OPTIONAL, default=default=]

The algorithm of the pipeline.

Integer apid [INPUT, OPTIONAL, default=No default value]

ApID of observation to be processed. Aspected value: 1030 , 1031

Long obsid [IN, OPTIONAL, default=No default value]

Alternative input for the observation to be processed. HifiTimelineProduct ObservationContext created on the fly

Boolean step [INPUT, OPTIONAL, default=No default value]

if the pipeline should be run step by step

ObservationContext obs [IO, OPTIONAL, default=(alternative to htp)]

No default value, null allowed. The Observation Context processed , in input Level 0, in Output Level 1

HifiTimelineProduct htp [IO, OPTIONAL, default=(alternative to obs)]

the HifiTimelineProduct to be processed.

Not Used cal [IN, MANDATORY, default=no default value]

type palStore [NOT Used, MANDATORY, default=no default value]

String db [IN, OPTIONAL, default=default= Configuration.getProperty("var.database.devel")]

the database containing the specified obsid.

Boolean gui [IN, OPTIONAL, default=default=False.]

only used for a test progress bar. TBD . If the progress bar is displayed or not.