The PACS Calibration Framework

A comprehensive guide

Rik Huygen • KULeuven • Instituut voor Sterrenkunde • rik.huygen@ster.kuleuven.be
Celestijnenlaan 200D bus 2401, 3001 Leuven, Belgium **T** +32 16 327043 **F** +32 16 327999

Issue 0.13

Document Change Record

Issue	Date	Comments
0.1	16 July 2010	First draft, document structure, gathering ideas, transfer text from existing package documentation, … This issue describes features available up to hcss.dp.pacs-5.0.650.
0.2	19 July 2010	Incorporate comments by Katrina Exter, started a glossary section.
0.3	28 July 2010	Incorporate comments by Cate Liu
0.4	8 Sep 2010	Added section on switching configurations in HIPE.
0.5	27 Sep 2010	Added information requested in PACS-1193 (Lack of documentation for cal product update wrt meta data). Changes in section 3.4.1. Meta Data of Calibration Products.
0.6	12 Oct 2010	Added section on calibration plugins for pipeline processing.
0.7	6 Dec 2010	Added documentation for new location of calibration products and Updater.
0.8	14 Dec 2010	Documented the new Updater that was implemented for PACS-2833. Added a section on calibration sets. The functionality is available as of CIB 6.0.1452.
0.9	17 Dec 2010	Added documentation for methods (see PACS-1167).
0.10	27 Jan 2011	Documented the changes in the Updater as a result of PACS-3306.
0.11	23-Feb-2011	Documented changes as a result of the following JIRA issues: PACS-2892, PACS-2868. Added user documentation for getCalProduct(). The expert user section on Updating Calibration Products has been updated. A new section on creating calibration products is started.
0.12	23-Mar-2011	Added information about the different ways to access a calTree, i.e. via the obs.calibration, or using getCalTree(obs=obs).
0.13	26-Jul-2011	Added a basic description of the Calibration Sets View.

Last modified by Rik Huygen on 26 Jul 2011 13:55.

Reference Documents

[R1]	HERSCHEL-HSC-DOC-0959	Herschel Products Definitions Document, issue 1.0, Feb 2010.
[R2]		PACS Calibration Products, TBW.

Rik Huygen • KULeuven • Instituut voor Sterrenkunde • rik.huygen@ster.kuleuven.be
Celestijnenlaan 200D bus 2401, 3001 Leuven, Belgium **T** +32 16 327043 **F** +32 16 327999

Table of Contents

1. Introduction

This document is a comprehensive manual of the PACS Calibration Framework for both users and developers.

The first section is the user manual. This section explains how the calibration framework is used by general users and calibration scientists from within HIPE. The language of choice in this section is Jython.

The second section is the expert user manual. This section explain how to interact with the framework as a calibration scientist. You will learn how to create new versions of calibration products with updated contents. This section also describes how to load and use calibration products for different instrument models. The language of choice for this section is Jython.

The third section is the developer manual. This section explains how the calibration framework is used in pipelines and by other developers. The section focuses on how you develop your code against the PACS calibration framework. The language of choice for this section is Java, with some examples in Jython.

The fourth section is a framework developer manual and is about developing the calibration framework itself. This section is highly specialized and should in principle be read by someone who is going to make changes to the calibration framework or someone who will be developing a new calibration product type. The language of choice for this section is Java.

Users are expected to have basic knowledge of Jython and understand the PACS instrument. In addition users need to understand the data structures (Product and Context) used in the Herschel Common Software System (HCSS), and how data items within these data structures are accessed from the command line and the GUIs. A basic knowledge of pools, storages and tasks is certainly a advantage.

Expert users are expected to have basic knowledge of cvs.

Developers are expected to know the Java language and understand the ins and outs of the Product Access Layer (PAL) in the HCSS.

1.1. Design requirements

This section describes the top level design requirements for the PACS calibration framework.

- it shall be easy to access the calibration products and their content. User should not have to apply difficult constructs in order to get to the data they need. If the design of the system makes the interface awkward, either the design is wrong or too complicated, or convenience functions must be provided.
- it shall be obvious for the user to see which calibration products are used during processing, where they come from and what their version is.
- it shall be easy to update the content of a calibration product. Updating a calibration product shall be as easy as assigning a value to a variable.
- calibration specialists shall be able to create and populate a new calibration product

- calibration products that are time dependent, i.e. a calibration evolves during the mission, shall be handled transparently by the framework
- calibration products should be available to the user at all times. That means the user should have the calibration products on his/her local system in a convenient location.

2. The User Manual

2.1. TBW

- How can a user know which version of a calibration product was used in which build, i.e. which is the first build where a certain calibration product was used, at what date?
- Viewing more information about calibration products, i.e. productNotes and versionNotes.
- fm.photometer.responsivity = getCalProduct("photometer", "responsivity", scope="TEST")
- Although scope BASE and TEST still exist they are not used anymore. pcal-community has no time dependency for TEST, pcal-icc TEST is identical to BASE.

2.2. Introduction

What is this so-called *calibration framework*? It is the generic name for all the software that provides the definition of the calibration products and the tools that are used to access the information in the calibration products.

Most of this software is written in Java and not of interest to the general user. However, intuitive and convenient jython functions are provided for users to access information from the calibration framework.

The starting point in this framework from a user perspective is the *calibration tree*, also known as *calTree*. The calibration tree is a simple tree-like structure which points to all the calibration products as leaves.

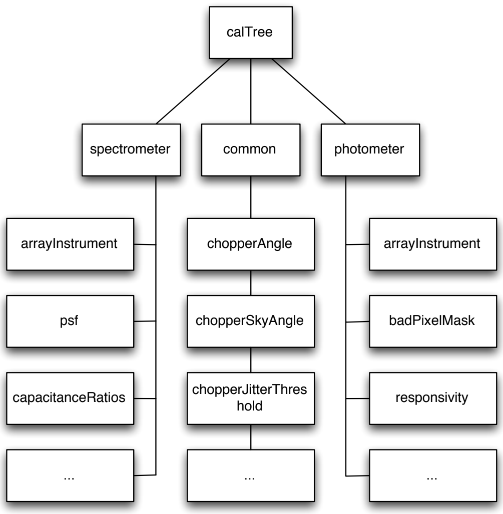
From an automatic processing point of view, as an example, the PACS relative spectral response is corrected by the pipeline task rsrfCal(..) which uses several calibration products called rsrfB2A, rsrfB2B, rsrfB3A and rsrfR1. This task will first load the calibration tree via the getCalTree(..) command and point the pipeline task to the correct calibration products on disk. Calibration products are only loaded when used by the pipeline task.

The calibration products, the calibration tree and the commands to access them are all part of the calibration framework.

2.3. The Calibration Tree

The calibration tree has three branches: common, spectrometer and photometer. The *spectrometer* and *photometer* branch each contain the calibration products that are specific for that instrument. The *common* branch contains all the calibration products which are shared between the different units in the instrument and those which do not fall easily within one of the two other branches. The branches correspond to the three sections on PACS calibration products as described in Section 4.2 of the *Herschel Product Definition* document [R1]. ThIs document is part of the on-line HIPE documentation.

The calTree is schematically pictured below.



The branches provide structure to the calibration tree and allows the same names to be used for equivalent calibration products in both the photometer and the spectrometer, e.g. ArrayInstrument in the above picture.

As of this writing there are 8 calibration products in the common branch, 38 for the spectrometer, and 23 for the photometer. You can get a list of all the calibration products in a branch by simply printing the branch from the command line:

```
HIPE> print calTree.photometer
PacsCalPhot Calibration Products:
  absorption                        : FM, 2
  arrayInstrument                   : FM, 6
  badPixelMask                      : FM, 5
  calSources                        : FM, 1
  clSaturationLimits                : FM, 1
  clTransferFunction                : FM, 1
  corrZeroLevel                     : FM, 3
  crosstalkMatrix                   : FM, 2
  detectorSortMatrix                : FM, 3
  diffCS                            : FM, 3
  filterTransmission                : FM, 1
  flatField                         : FM, 3
  gain                              : FM, 1
  invntt                            : FM, 1
  invnttBL                          : FM, 2
  invnttBS                          : FM, 2
  invnttRed                         : FM, 2
  masks                             : FM, 1
  noisePerPixel                     : FM, 1
  photometricStabilityThreshold     : FM, 1
  responsivity                      : FM, 5
  satLimits                         : FM, 2
  subArrayArray                     : FM, 5
  timedep                           : FM, 6
```

The FM in this table stands for the Flight Model of the PACS instrument and the integer number is the version number of the calibration product. Both concepts are explained in more detail below.

The user can access these products and their content using the simple dot-notation to navigate from the root of the calibration tree down to each value in each product. As an example, to navigate to the responsivity calibration product from the calTree you type:

```
HIPE> resp = calTree.photometer.responsivity
```

So how do I get hold of that root of the calibration tree? The easiest way is when you start from an observation, each Herschel observation has an associated calibration tree. That tree is specific for that observation and the software by which the observation was processed. The tree contains all calibration products that are or will be used to process the observation. It does not necessarily contain the latest versions of all calibration products. You can get access to it with the following syntax (`obs` is the observation that is accessible from within your session):

```
HIPE> calTree = obs.calibration
```

If you do not have an observation to process but still need access to the calibration tree, you will have to load a calibration tree into your session. The framework provides a simple command to do this:

```
HIPE> calTree = getCalTree()
```

This will load the default calibration tree into the `calTree` variable. The default calTree is the latest version of the calibration tree for the current instrument model and the *baseline* scope. These concepts are explained in the following sections.

The preferred way to get a calibration tree into your session is however to use getCalTree() with an argument, preferably the observation time or the time of a frame.

```
HIPE> calTree = getCalTree(obs=obs)
```

```
HIPE> calTree = getCalTree(time=frame.startTime)
```

This will load the proper calibration products and make sure time dependent products are chosen for the given time.

So what is the difference between the following two statements?

```
calTree = obs.calibration
```

```
calTree = getCalTree(obs=obs)
```

There is quite a big difference in the versions of the calibration products that you will get back. The first line returns the calibration tree that was used during processing of that observation with SPG (provided that you didn't change anything to the observation after you extracted it from the HSA). This calibration tree is the version which was active for the SPG version at the time of processing. If you want to reprocess the observation the same way as done by SPG **at that time**, this is what you need.

The second line returns the latest and most up-to-date calibration tree from your local machine (make sure the updater was used). The obs argument is used for time dependent calibration products, currently only Siam, where the time of the observation is used to select the correct version of the time dependent calibration product. This is what you want to use to reprocess an observation with the latest knowledge in PACS calibration.

2.3.1. Instrument Models

Since we are operating different instrument models during the mission lifetime, we also have for each model different calibrations. A calibration tree is therefore identified by its model.

```
HIPE> print calTree.modelName
FM
```

In the printout of the photometer calibration branch in the previous section, you already saw the model showing up in the second column of the printout. The framework currently knows about three different models, i.e. the *flight model* (FM or Flight) which is the current instrument model on board the Herschel satellite, the *flight spare* (FS) which is a scaled down model of the instrument (in terms of number of detectors ...) that we operate in the laboratory at MPE to test on-board software updates and investigate instrument anomalies. There is also the *qualification model* (QM) which was used in a previous phase in the mission to qualify the instrument design and prove the mission goals.

You can load a calibration tree for a specific model if you specify the *model* keyword as an argument to getCalTree.

```
HIPE> calTree = getCalTree(model="FM")
```

The flight model (FM) is the default model for the calibration tree. That means this model is used when nothing is specified in the getCalTree() command. The pipeline processing is also done with this default instrument model.

Please note that the HIPE installation only contains the calibration tree for the flight model. Requesting the calTree for a different model will result in a IllegalArgumentException telling you that no calibration tree can be found for the given model.

```
HIPE> calTree = getCalTree(model="FS")
java.lang.IllegalArgumentException: No calibration tree available for the given model and
scope [model=FS, scope=BASE].
```

Loading a calibration tree for a different model is explained in the expert user section.

2.3.2. Different Scopes for Calibration Trees

Calibrating an instrument is a difficult and time consuming task. It is also one of the major activities of the instrument control centers (ICC). If therefore a certain aspect of the instrument calibration needs to be improved, the calibration scientists in the ICC are working with improved calibration products in order to test and validate them. We say that the ICC experts are working in a different *scope* as any other user. This scope is called TEST and it defines a calibration tree that contains products that are not yet validated by the ICC. Outside the ICC, users are working in what is called the *baseline* calibration and the scope that defines this baseline is called BASE[1]. The baseline calibration tree contains all the products that have been tested, validated and released by the ICC.

You can specify the scope of the calTree as a keyword argument:

```
HIPE> calTree = getCalTree(scope="BASE")
```

BASE is the default scope that is used when no scope is specified when loading a calibration tree. BASE is also the scope used by the pipeline processing. There is a third scope defined in the framework which is used for bulk reprocessing. This is further described in the expert user section.

You can check the scope for your calTree as follows:

```
HIPE> print calTree.scope
BASE
```

2.3.3. Versions of Calibration Trees

Since calibration products are updated at regular intervals during the mission, we need to version control the sets of calibration products that are used by data processing, both in the pipeline and interactively. This is needed in order to trace the changes to calibration products and also to be able to reproduce data processing results.

You can check the version for your calTree as follows:

```
HIPE> print calTree.version
6
```

If for some reason you need to access a previous version of the calTree, you can specify the version number as an argument when loading the calTree:

```
HIPE> calTree_v3 = getCalTree(version=3)
```

The version of the calibration tree is not the same as the version of the calibration products that were listed in the beginning of this chapter. The relation between the calTree version and the individual calibration product versions is explained in the expert user section.

[1] Note however that the TEST scope is still available for users outside the ICC, but is not defined as the default.

A full explanation of the getCalTree() command is given in appendix A: User Commands.

2.3.4. Calibration Sets

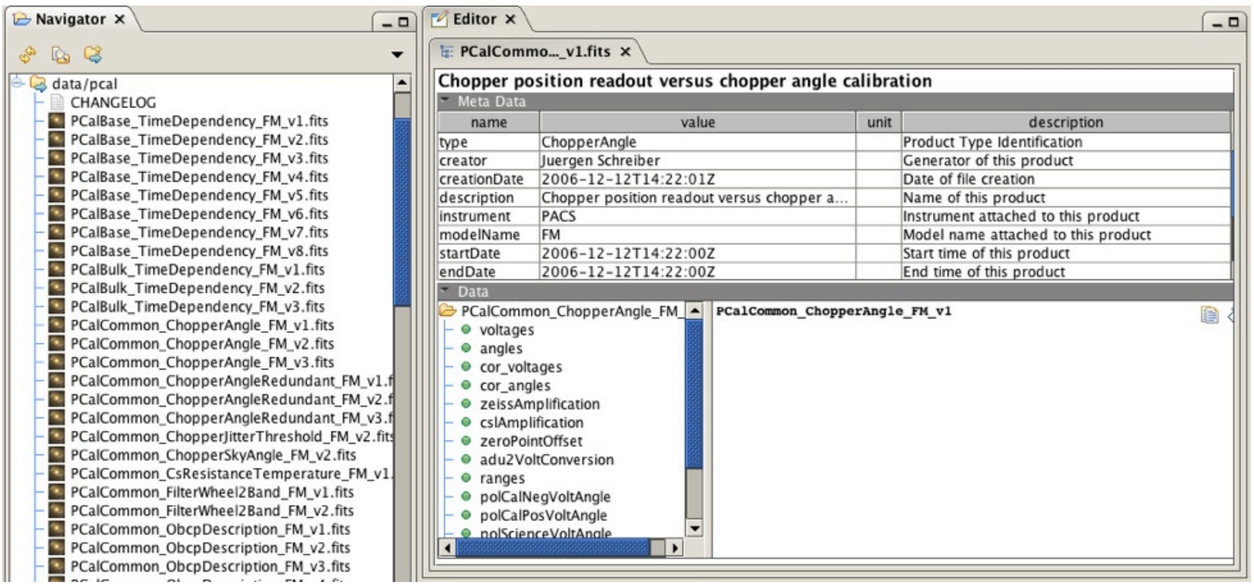
A Calibration set is the group of all calibration products that together form a unique collection. Any reference to a calibration set unambiguously defines the versions of all calibration products that are comprised in that set, irrespective of the scope of the calibration tree and the time of the observation. The version of a calibration set is determined from the time dependency product. You can inspect the calibration sets that are installed on your machine by opening the View for Calibration Sets in HIPE. This View is located in the **Window -> Show View -> Workbench** menu. The View is explain in detail in the section on Inspecting Calibration Sets and Products.

2.4. Calibration Products

Calibration products are by default located in the directory[2] `${user.home}/.hcss/data/pcal-community`. This directory contains all calibration products for the flight model in different versions. Older versions are there for reference and to allow you to compare results while processing your observations with different sets of calibration product versions.

The calibration products are stored as FITS files with a strict naming convention. The naming is explained further down in the section on build a FITS file name for your calibration product.

In the HIPE Navigator a double click on any of the calibration products will open the Product Viewer showing the structure and content of the product. In this view you can easily navigate through your calibration products to inspect their contents. I recommend that you set up a user area in the HIPE Navigator which points directly to the location of your calibration products. User areas can be managed in the Preference panel, a shortcut is available in the toolbar of the Navigator. Refer to the general HIPE documentation for further information on the Navigator. The structure and contents of the calibration products is explained in detail in a separate document PACS Calibration Products.



```
data/pcal
  CHANGELOG
  PCalBase_TimeDependency_FM_v1.fits
  PCalBase_TimeDependency_FM_v2.fits
  PCalBase_TimeDependency_FM_v3.fits
  PCalBase_TimeDependency_FM_v4.fits
  PCalBase_TimeDependency_FM_v5.fits
  PCalBase_TimeDependency_FM_v6.fits
  PCalBase_TimeDependency_FM_v7.fits
  PCalBase_TimeDependency_FM_v8.fits
  PCalBulk_TimeDependency_FM_v1.fits
  PCalBulk_TimeDependency_FM_v2.fits
  PCalBulk_TimeDependency_FM_v3.fits
  PCalCommon_ChopperAngle_FM_v1.fits
  PCalCommon_ChopperAngle_FM_v2.fits
  PCalCommon_ChopperAngle_FM_v3.fits
  PCalCommon_ChopperAngleRedundant_FM_v1.f
  PCalCommon_ChopperAngleRedundant_FM_v2.f
  PCalCommon_ChopperAngleRedundant_FM_v3.f
  PCalCommon_ChopperJitterThreshold_FM_v2.fits
  PCalCommon_ChopperSkyAngle_FM_v2.fits
  PCalCommon_CsResistanceTemperature_FM_v1.
  PCalCommon_FilterWheel2Band_FM_v1.fits
  PCalCommon_FilterWheel2Band_FM_v2.fits
  PCalCommon_ObcpDescription_FM_v1.fits
  PCalCommon_ObcpDescription_FM_v2.fits
  PCalCommon_ObcpDescription_FM_v3.fits
```

Chopper position readout versus chopper angle calibration

Meta Data

name	value	unit	description
type	ChopperAngle		Product Type Identification
creator	Juergen Schreiber		Generator of this product
creationDate	2006-12-12T14:22:01Z		Date of file creation
description	Chopper position readout versus chopper a...		Name of this product
instrument	PACS		Instrument attached to this product
modelName	FM		Model name attached to this product
startDate	2006-12-12T14:22:00Z		Start time of this product
endDate	2006-12-12T14:22:00Z		End time of this product

Data

```
PCalCommon_ChopperAngle_FM_        PCalCommon_ChopperAngle_FM_v1
  voltages
  angles
  cor_voltages
  cor_angles
  zeissAmplification
  cslAmplification
  zeroPointOffset
  adu2VoltConversion
  ranges
  polCalNegVoltAngle
  polCalPosVoltAngle
  polScienceVoltAngle
```

As explained above versions of calibration products are related to the version of the calibration tree and the current scope of that tree. Changing the version of a calibration product on the current tree is explained further down in this section.

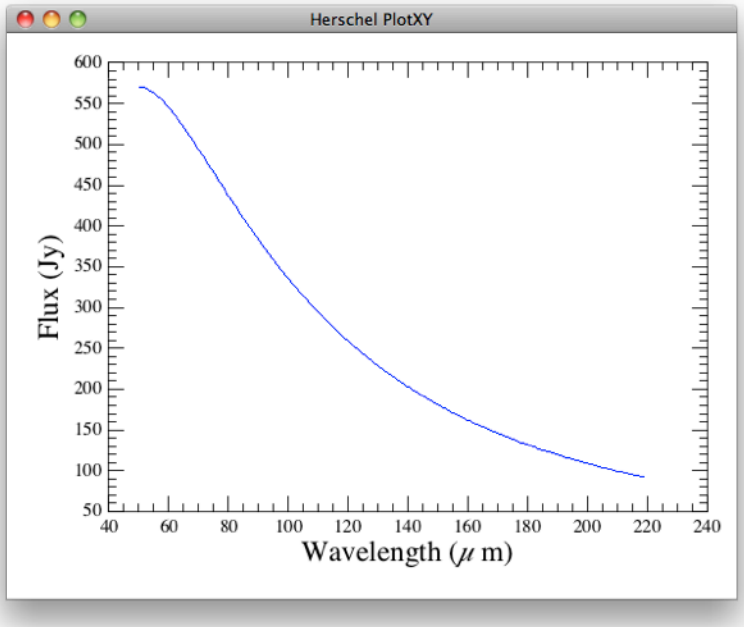
Accessing information from the calibration products is done by not-navigation from the calTree to the calibration product and further down to the individual values, tables and numeric arrays. Note that whenever you access a calibration

[2] to be correct, by default the location for the calibration products is determined from the property var.hcss.cfg.dir, so if you didn't touch this property, the location will be as stated in the text, i.e. `.hcss` in your home directory.

product, it is only loaded into memory at that time. The reason is to save memory consumption when not all products are accessed as there are some quite large calibration products defined for PACS (an example are the relative spectral response functions, RSRF).

As an example, the commands below extract the telescope background from the calibration product and show a plot of the flux per wavelength.

```
telescopeBackground = calTree.spectrometer.telescopeBackground
plot = PlotXY(telescopeBackground.wavelengths, telescopeBackground.fluxJy)
plot.xaxis.titleText = "Wavelength ($\micro$m)"
plot.yaxis.titleText = "Flux (Jy)"
```



If you need a short overview of the commands to use for accessing this calibration data, you can execute the methods command on each of the calibration products. This will show you a list of methods you can use with their parameter types and the type of the return value. As an example, for the telescopeBackground used above, the following is printed:

```
HIPE>print calTree.spectrometer.telescopeBackground.methods
getFluxJy() returns Double1d
getFluxWatt() returns Double1d
getWavelengths() returns Double1d
setFluxJy(Double1d)
setFluxWatt(Double1d)
setWavelengths(Double1d)
```

These are Java methods, but as you probably know, any method that starts with get and doesn't take arguments can be abbreviated in Jython, e.g. `getFluxJy()` becomes `fluxJy`. All these convenience methods for calibration products are explained in great detail in the PACS Calibration Products document.

TBW Creating and maintaining calibration products is explained in the expert user manual chapter.

2.4.1. Time Dependency of Calibration Products

A number of calibration products have limited validity because of instrument evolution and aging. Especially those calibration products that characterize mechanisms in an instrument such as the characterization of the grating and the wavelength calibration. These products are called time dependent calibration products.

The validity of time dependent calibration products is defined in a time dependency look-up table which contains for each such product the time period for which it is valid and can be safely used in data processing.

The calibration framework knows about this time dependency table and uses it to lookup the correct version of a calibration product based on the time of the observation or a time given as an argument to the getCalTree(..) command.

```
HIPE> calTree = getCalTree(obs=myObservation)
HIPE> calTree = getCalTree(time="17-Oct-2004 13:42:00")
```

In the `obs` argument you pass an observation to the calibration tree which will use the start time of the observation to lookup the correct version of a calibration product.

The `time` argument can be either ASCII text in the format "dd-MMM-yyyy HH:mm:ss", or a FineTime. The latter can be used e.g. by passing the start date of a Frames product. As any product in the system has a start and end time, you can easily get a valid calTree as follows:

```
HIPE> calTree = getCalTree(scope="TEST", time=frames.startDate)
```

2.4.2. Accessing versions of calibration products

During your analysis you might want to access another version of a calibration product then the version available from the calTree. The framework provides a convenience function to access a particular version of a calibration product. The function getCalProduct(...) returns a calibration product of the requested type and version. The function takes three arguments: unit, name, and version.

The following example loads version 1 of the ArrayInstrument product for the spectrometer.

```
  HIPE> ai = getCalProduct("spectrometer", "arrayInstrument", 1)
```

Once you have loaded a specific version of a calibration product, it is very easy to use this version in your pipeline processing. You can either directly pass the calibration product to the pipeline step, or you can update the calibration tree in memory with this product. The first example uses the calibration product directly

```
HIPE> frames = specAssignRaDec(frames, arrayInstrument=ai)
```

The next example below replaces the current calTree version of ArrayInstrument with the version 1 loaded above:

```
HIPE> calTree = getCalTree()
HIPE> calTree.spectrometer.arrayInstrument = ai
```

Or make the change in a one-liner

```
HIPE> calTree.spectrometer.arrayInstrument = \
          getCalProduct("spectrometer", "arrayInstrument", 1)
```

You can now pass the calTree into the pipeline steps, and version 1 of ArrayInstrument will be used.

```
HIPE> frames = specAssignRaDec(frames, calTree=calTree)
```

If you do not specify the version number of the calibration product in getCalProduct(..) the latest available version will be returned.

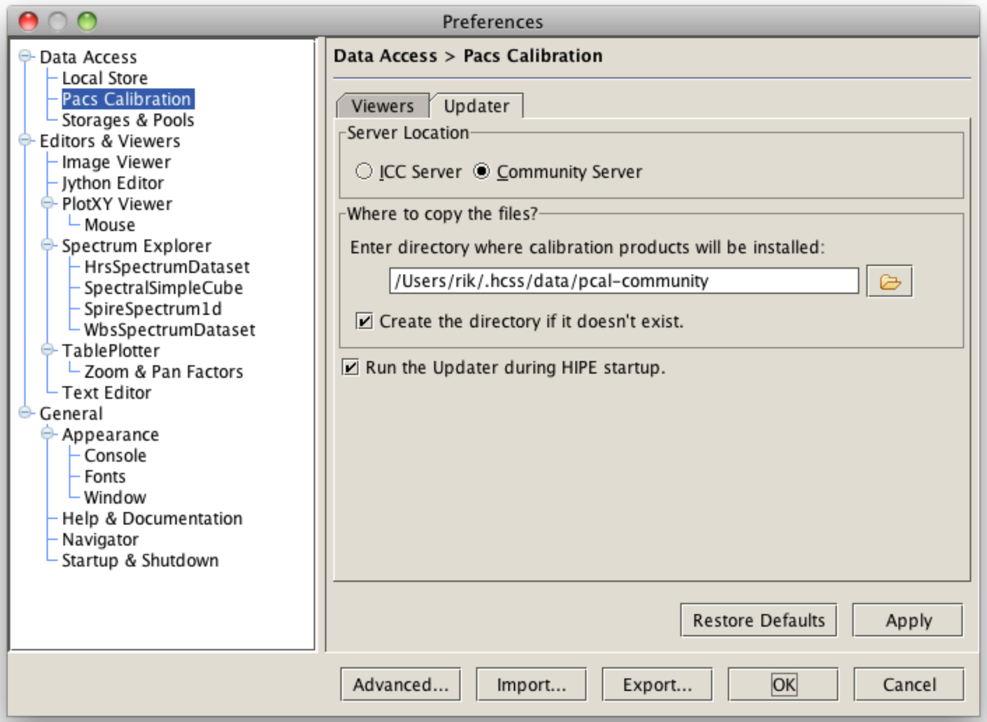
Sometimes you just want to load the latest calibration product. If you do not specify a version to getCalProduct(), the latest version for the default scope will be returned. The default scope is BASE. If you want the latest version for a different scope, you can specify the scope as an optional keyword argument.

```
HIPE> ai = getCalProduct("spectrometer", "arrayInstrument", scope="BULK")
```

Other optional arguments to getCalProduct() are `model="FM"` and `verbose=True`. The getCalProduct() command is described in full detail in the Appendix.

2.5. Configuring the Calibration Framework

There is no need to configure the calibration framework on a user level. The latest calibration products are downloaded with the Updater to a default location and the framework is configured to access these calibration products from that known location. Nevertheless, the user has the possibility to configure a few preferences. In the Preferences panel these settings can be found under Data Access > Pacs Calibration. For the Updater, the preference panel is shown below. By default the community server is selected and calibration products will be downloaded to the `${user.home}/.hcss/data/pcal-community` directory. If you are an ICC member and need to test the non-consolidated calibration products,



Server Location
ICC Server [✓ Community Server]

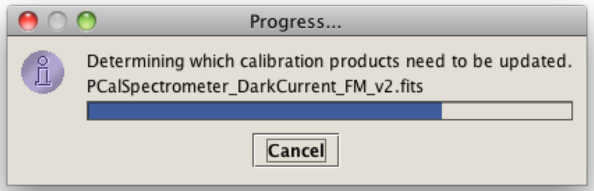
Where to copy the files?
Enter directory where calibration products will be installed:
/Users/rik/.hcss/data/pcal-community
☑ Create the directory if it doesn't exist.

☑ Run the Updater during HIPE startup.

you can select the ICC Server in the preferences. The Updater will then ask for a username and password before downloading any calibration products. If there is a need to access calibration products from another location or from the HSA, read the corresponding section on configuring the framework in the expert user chapter below.

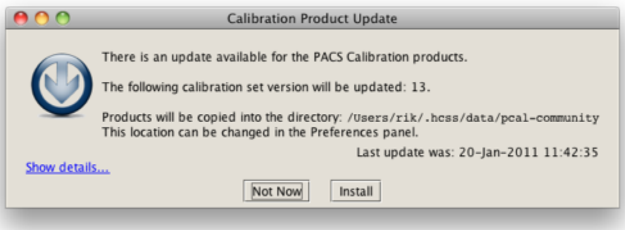
2.6. Updating the Calibration Products

Calibration products need to be installed on your local disk when using HIPE for the first time. Installation is a simple process that just takes a few minutes when you have a fast internet connection. Go to the Tools > pacs-cal menu and select run Updater... to start the update process. This will show a progress dialog when analyzing which calibration



Determining which calibration products need to be updated.
PCalSpectrometer_DarkCurrent_FM_v2.fits

products need to be updated, and finally come up with a dialog with details about the update that will be performed. At any point you can decide to abort the update or installation of any calibration product by clicking the Cancel button.



Calibration Product Update

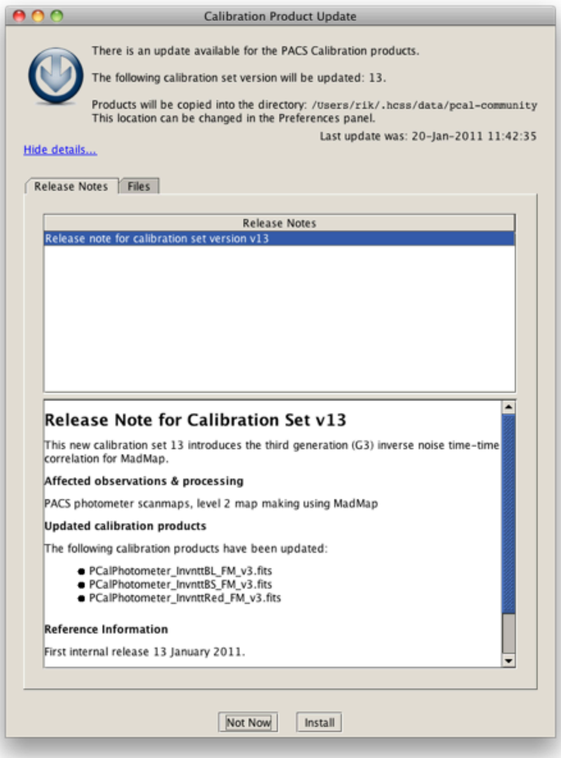
There is an update available for the PACS Calibration products.

The following calibration set version will be updated: 13.

Products will be copied into the directory: `/Users/rik/.hcss/data/pcal-community`
This location can be changed in the Preferences panel.

Last update was: 20-Jan-2011 11:42:35

The dialog at the left shows the basic information about the required update. It informs you about the calibration sets that need an update and where the products will be installed. You can read about calibration sets in the section on the Calibration Tree. In the lower right corner there is the date and time of the last update that you performed (in the example that was 20th of january). When you press the **Show Details…** link, the dialog expands to show the release notes and the list of all calibration products that need an update. By default the release note for the last calibration set version is selected and shown. The release notes will give you an explanation of the changes with respect to the previous update. This will allow you decide if you want to proceed with the update or not. One reason to cancel the update is because you are in the middle of a data reduction process for several observations and you do not want to change the calibration in order to have a consistent result. If you click on the Files tab, all the individual calibration products that will be updated are listed. The size of the products will give you an indication of how long the update might take depending on your network speed. When you click on a calibration product in the table, a release note for that product is shown in the lower panel when available. The **Installed** column shows if the product is installed on your system. This column will be automatically updated during the installation process.



Calibration Product Update

There is an update available for the PACS Calibration products.

The following calibration set version will be updated: 13.

Products will be copied into the directory: `/Users/rik/.hcss/data/pcal-community`
This location can be changed in the Preferences panel.

Last update was: 20-Jan-2011 11:42:35

Release Notes
Release note for calibration set version v13

Release Note for Calibration Set v13

This new calibration set 13 introduces the third generation (G3) inverse noise time-time correlation for MadMap.

Affected observations & processing

PACS photometer scanmaps, level 2 map making using MadMap

Updated calibration products

The following calibration products have been updated:

- PCalPhotometer_InvnttBL_FM_v3.fits
- PCalPhotometer_InvnttBS_FM_v3.fits
- PCalPhotometer_InvnttRed_FM_v3.fits

Reference Information

First internal release 13 January 2011.

The Calibration Product Update is started automatically in the background when you start up HIPE. This is to ensure you are at least notified when there is an update of a calibration product. The Updater can be started manually at any time from the **Tools** > **pacs-cal** menu and will then also run in the background without blocking your usual HIPE session.

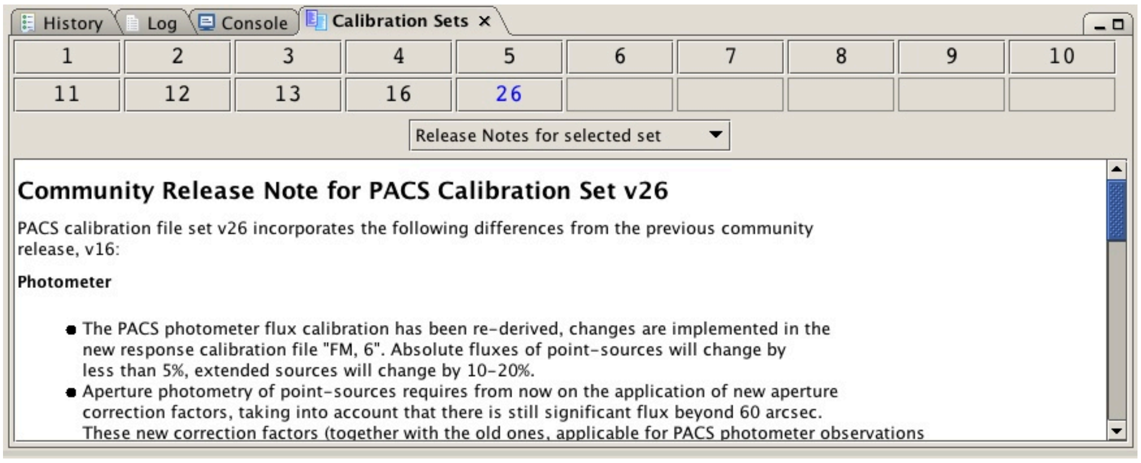
You can still abort the installation process by pressing the **Not Now** button. If you press the **Install** button the Updater will start to download the calibration products and the **Not Now** button will change into a **Stop** button. When you press the **Stop** button during the download, the process will stop only after it properly installed the product it is currently downloading. Be aware that this might leave your calibration setup in an inconsistent state as not all calibration products will be available on your local machine. You will need to start the Updater again at a later time to finish the update and put your system back into a consistent state.

If no updates are required, a dialog will popup to notify you that there are no updates available.

2.7. Inspecting your Calibration Sets and Products

2.7.1. The Calibration Sets View

The calibration sets View allows you to inspect the calibration sets that have been installed on your system. The view shows the different versions and their release notes and individual products. The View can be opened from the menu item **Window -> Show View -> Workbench**. The figures below show the calibration sets view with set 26 selected. By default, the view will show the release note for the selected set. You can change that by selecting the file list from the drop down menu under the set numbers.

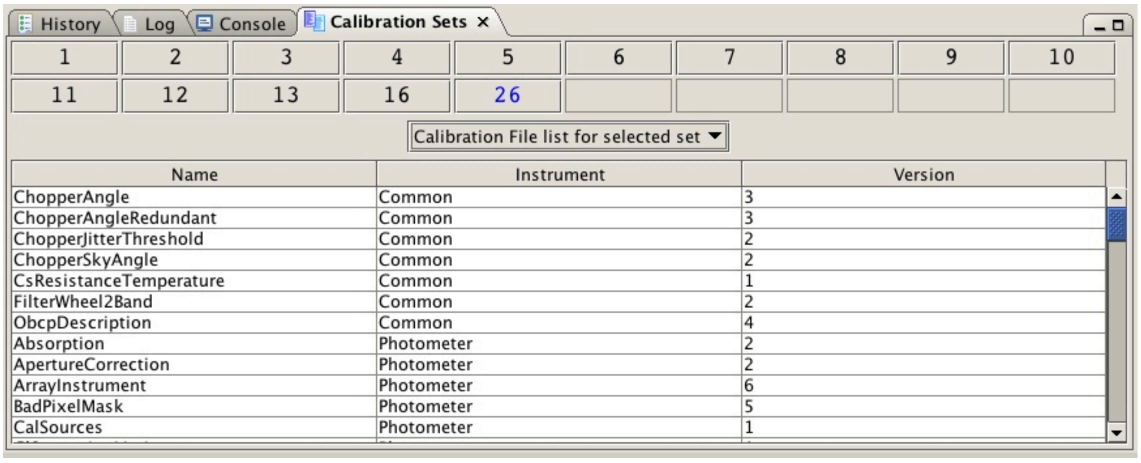


Community Release Note for PACS Calibration Set v26

PACS calibration file set v26 incorporates the following differences from the previous community release, v16:

Photometer

- The PACS photometer flux calibration has been re-derived, changes are implemented in the new response calibration file "FM, 6". Absolute fluxes of point-sources will change by less than 5%, extended sources will change by 10-20%.
- Aperture photometry of point-sources requires from now on the application of new aperture correction factors, taking into account that there is still significant flux beyond 60 arcsec. These new correction factors (together with the old ones, applicable for PACS photometer observations



Name	Instrument	Version
ChopperAngle	Common	3
ChopperAngleRedundant	Common	3
ChopperJitterThreshold	Common	2
ChopperSkyAngle	Common	2
CsResistanceTemperature	Common	1
FilterWheel2Band	Common	2
ObcpDescription	Common	4
Absorption	Photometer	2
ApertureCorrection	Photometer	2
ArrayInstrument	Photometer	6
BadPixelMask	Photometer	5
CalSources	Photometer	1

The release note provides all information on the changes to calibration products with respect to the previously released calibration set. In the example above, the release note for set 26 describes in detail the changes that were made since set 16.

Please note that not all changes to calibration products are immediately available to the astronomical community. In this case there were ten internal ICC releases before the complete set was validated by the Herschel Science Center (HSC) and released to the community. We like to keep the version numbers of the sets consistent with the ICC versions in order to avoid confusion when problems are reported and we need to go back and investigate.

3. The Expert User Manual

3.1. TBW

Explain BULK scope and reprocessing

Explain loading a calTree for a different model, where to get these cal products and how to install them on you system.

Explain utilities provided by the framework to make the life of the expert user easier.

3.2. Introduction

This section is for expert users of the calibration framework. Expert users are calibration scientists who have to update the content of the calibration products, then test, validate and release new versions of calibration products. Expert users are also those who operate the pipeline processing and test pipeline deliveries as they need to know about different scopes, in particular the BULK scope.

3.3. Loading a calTree for a different model

The calibration framework that is distributed and included in the installation on your local system contains the calibration products for the flight model (FM). If you need calibration products from other instrument models, you will need to install those products on your local system and configure the framework to point to these calibration products. Then you need to load the calTree for that model. The different steps are explained below.

3.3.1. Download and install calibration products

TBW Calibration products can be installed by a HIPE plugin.

3.3.2. Reconfigure your framework

TBW

3.3.3. Load the calTree

3.4. Calibration Products

TBW Include example of a calibration product, how the meta data looks like, in a viewer and from the command line, maybe color different parts of the example and explain them in more detail in the text, ….

3.4.1. Meta Data of Calibration Products

All calibration products use a set of common meta data. The framework expects that these meta data are populated correctly and consistently. There is a utility in the package to check this consistency, use it whenever you make changes to the calibration products (see Running the Calibration File Checker below).

All meta data can be set by a convenient assignment statement using the name of the meta data keyword at the left-hand side of the assignment. An example is given for each meta data keyword below (`cal` is the variable that contains you calibration product in your session).

author

The author of the product, i.e. the one that provides its contents, not its format. The parameter is of type String and can be set as follows:

```
cal.author = "Rik Huygen"
```

where cal is the calibration product.

calFileId

The name of the calibration product. This is also the class file for this product. This shall be equal to what is set in the type keyword (see below). The parameter is of type String and can be set as follows:

```
cal.calFileId = "ChopperAngle"
```

where cal is the calibration product. The name of the calibration product must correspond exactly to the name of the Java class that represents the product. For the example above, the class name can be printed as follows:

```
HIPE> print cal.__class__
herschel.pacs.cal.common.ChopperAngle
```

The calFileId must be set to the last part of this fully qualified name.

calFileVersion

The version of the calibration product with respect to its contents. This number is usually defined by the author whenever a change is made to the content of the calibration product. The calFileVersion is also reflected in the fileName. This version number must be an integer and is usually just increasing with every new version of the product. It can be set as follows:

```
cal.calFileVersion = 3
```

formatVersion

This version number changes when the format of the calibration product changes, e.g. when a column is added to a table. A change in formatVersion is usually a change which needs to be done with care. It probably means the Java code of the product has to be adapted to work with the new format and stay backward compatible with all previous formats. Please check with a developer if the Java code needs to be adapted. The formatVersion is of type String and can be changed as follows:

```
cal.formatVersion = "2.3"
```

There is currently no guideline or convention for the major and minor format version. It is left to the developer and calibration scientist to decide when it is appropriate to change the major format version instead of the minor number.

fileName

The FITS filename for this calibration product. Filename conventions are explained in the section Build a FITS file name for your calibration product. Please note that the .fits extension must also be in the meta data. The fileName is of type String can be set as follows:

```
cal.fileName = "PCalCommon_ChopperAngle_FM_v1.fits"
```

These were the meta data parameters that are specific for the calibration framework of PACS. Then there are some attributes that need to be set for each Product:

creationDate

the time at which the calibration product was created. creationDate is of type FineTime and can be set as follows:

```
cal.creationDate = FineTime(Date())
```

creator

The person or process that created the product. This is usually different from the author because the creator is the person or process actually making the product while the author provides the contents (usually after extensive data processing). This parameter is of type String and can be set as follows:

```
cal.creator = "The name of an application, script or a person"
cal.creator = "hcss.dp.pacs-5.0.593"
```

description

this parameter is usually set by the constructor and doesn't need to be changed. description is of type String and if needed can be set as follows:

```
cal.description = "one-liner describing the calibration product"
```

endDate

see entry for **startDate** further down.

instrument

The name of the instrument, i.e. PacsConfig.PACS. The parameter is of type String and can be set as follows:

```
cal.instrument = PacsConfig.PACS
```

modelName

the instrument model name, i.e. FM, FS or QM. To be really on the save side, it is recommended to use the PacsConfig.QM, PacsConfig.FM, and PacsConfig.FS to set this meta data, e.g.

```
cal.modelName = PacsConfig.FM
```

startDate

the start and end date are mandatory meta data that are currently not used for calibration products. The start and end date are usually set equal to the creation date of the product. Note also that the start and end date have no meaning for time dependent products. The time ranges for time dependent calibration products are defined in the TimeDependency products that are delivered together with the calibration products. startDate and endDate are of type FineTime and can be set as follows:

```
cal.startDate = FineTime(Date())
```

type

The type (class name) of the calibration product. This is the same as in the calFileId keyword (redundancy, I know...), but interpreted by different parts of the system. The parameter is of type String and can be set as follows:

```
cal.type = "ChopperAngle"
```

3.4.2. Naming Convention for Calibration Products

The names for calibration products follow the standard naming convention for Java classes, i.e. CamelCase[※]. That means each word in the name starts with a capital letter (is capitalized). In addition, the product name should not include an indication of the instrument unit it is intended for, that is implicit because the product is attached to a calibration tree branch. The table below presents a few good and bad practice examples:

Calibration Product name	Comment	Correct name
ChopperSkyAngle	GOOD	
SpecBadPixelsMask	BAD, contains indication of sub unit spectrometer	BadPixelMask
PhotSortMatrix	BAD, contains indication of sub unit photometer	DetectorSortMatrix
SubarrayArray	BAD, not all words are capitalized	SubArrayArray
OBCPDescription	BAD, acronyms should also follow the CamelCase convention	ObcpDescription
FilterWheel2Band	GOOD	
BoloCSs	BAD, it is not clear from the name what this is and it contains an indication of the sub unit	CalSources

You have to see this naming as part of the framework. The calibration products do not stand by themselves, but are embedded in a structure or hierarchy which makes the context clear. Within the calibration tree structure, for example, there is a DetectorSortMatrix in both the photometer and the spectrometer branch. It is therefore not needed to include this information in the name of the product. The example code below should make this clear. The first line shows what a call to the conversion matrix for the blue channel would look like for the photometer if we do not use the naming convention. There is duplication of information, even twice. The second command line shows the same functionality, but now following the guidelines for naming convention.

```
HIPE> calTree = getCalTree()
HIPE> print calTree.photometer.photSortMatrix.photSortBlue    # DUPLICATION
HIPE> print calTree.photometer.detectorSortMatrix.blue        # NO DUPLICATION
```

Please note the first letter of the class names in the above examples is not upper case as you might have expected. The class name itself is however still CamelCase※, only it is used with a lower case first letter in the dot-navigation※ examples above. This case conversion is done automatically by jython.

3.4.3. Build a FITS file name for your calibration product

Now that you know how to name calibration products, the step to building a FITS file name for your calibration product is easy. The FITS name is constructed from all the parts that you already know about the product and that make the product unique, i.e. the branch, the product name or class name, the model name, and the version number. The format of a FITS file name is as follows:

```
PCal<Branch>_<ProductName>_<MODEL NAME>_v<version number>.fits
```

where:

`<Branch>` equals Common, Photometer, or Spectrometer,
`<ProductName>` is the class name of the calibration product,
`<MODEL NAME>` equals QM, FM, or FS, and
`<version number>` is the integer version assigned by the author of the product. This version number is the same as in the calFileVersion meta data keyword.

Time dependency products follows the same convention as normal calibration products. The branch is then replaced by the scope, i.e. Base, Test or Bulk, and the product name is always TimeDependency as in

```
PCalBase_TimeDependency_FM_v2.fits
```

The FITS file name must be put into the fileName meta data keyword, including the .fits extension.

3.5. Making Updates to Calibration Products

Whenever a change is made to a calibration product, the version number must be incremented. This means that both the calFileVersion and the corresponding number in the fileName meta data has to be incremented. The reason for this version number increase is reproducibility and traceability. We need to be able to reproduce any result that has been processed by any version of a calibration product, and we need to be able to track the changes made to each of the calibration products during the entire mission.

Each change must also be described in the versionNotes meta data. This is a text parameter that can hold xhtml tags for proper formatting in a GUI. The versionNotes usually describe briefly what the major change was and references the JIRA issue that describes the change in full detail.

The steps to follow in making updates to an existing calibration product are the following:

1. get hold of the latest applicable calibration product for which you want to update the content,
2. make the changes by using the access methods,
3. test the changes by running the appropriate pipeline steps or modules,
4. update the meta data to reflect the change in the calibration product,
5. save the calibration product as a FITS file and save it into CVS.

We will take a simple example and make a change to the sky angle for the chopper. The jython code below shows how to get the ChopperSkyAngle calibration product into the variable named csa. The third line applies the change by assigning a new value to the conversionFactor. Next we need to put back the calibration product into the common area of the calibration tree.

This last step is important because the calibration tree keeps no record of the calibration products it handed out. This is for memory reasons, because if the calibration tree would remember all calibration products the memory would fill up immediately by requesting huge products like the Rsrf products.

```
fm = getCalTree("FM")
csa = fm.common.chopperSkyAngle
csa.conversionFactor = 3.0
fm.common.chopperSkyAngle = csa
```

We now have a calibration tree which contains an updated version of the ChopperSkyAngle product in memory. We can either use the calibration product itself or the tree as an argument in our pipeline step in order to test the change, e.g.

```
outFrames = convertChopper2Angle(inFrames, chopperSkyAngle=csa)
outFrames = convertChopper2Angle(inFrames, calTree=fm)
```

Now if we want this change implemented in a new version of the calibration product, we need to update a number of meta data entries like calFileVersion and the fileName. Additionally, the versionNotes shall be updated with information specific to this version.

You can find a complete list of meta data in the overview in the section on meta data of calibration products. For this simple update of the calibration product contents, the following meta data needs to be changed:

- calFileVersion: increase this number by one indicating a new version of this product
- fileName: the fileName carries the version number also, so change the fileName that it properly reflects the correct version
- versionNotes: the information should be accurate and to the point. Only comment the changes for this particular version and provide the JIRA issue number that requested the change.

Inspect all your changes once again from either the command line by printing the values and meta data or by using the Product Viewer. If you find everything is as expected, save the updated product in a FITS file with FitsArchive. The FITS filename must be the same as what is in the fileName meta data.

```
FitsArchive().save("/tmp/" + csa.fileName, csa)
```

To make your changes available in the build, you first need to checkout the CVS module containing all calibration products. The cvs module name is `pacs_data_pcal`, so after you have logged into cvs and set the CVSROOT, you can checkout the calibration products (beware these commands are on the UNIX prompt):

```
cvs co pacs_data_pcal
```

This will create a `develop` directory with the calibration products in `develop/data/pacs/data/pcal`. Copy your updated product into this directory. Then update the CHANGELOG file which is located in the same directory as the calibration products. After this you can commit and tag the cvs module. Please change the lines below for your particular needs, the filename, version, comment and D-tag below are just example entries.

```
cd develop/data/pacs/data/pcal
cvs add PCalCommon_ChopperSkyAngle_FM_v3.fits
cd -
cvs commit -m "updated ChopperSkyAngle to version 3"
cvs tag D_PACS_DATA_PCAL_0_142
```

This will deliver your calibration product to the system. Now please notify the person responsible for updating the web server that is used by the automatic update process in HIPE. This person is currently rik.huygen@ster.kuleuven.be.

3.6. Creating a new Calibration Product

Creating a new calibration product currently requires someone with Java skills. Although the complete preparation for a new product can be done in Jython from the HIPE command line, the final step is the construction of a Java class to hold the calibration product. The reason for this is that calibration products are kept in a Product pool which can only work with Java classes.

The steps to follow in creating a new cal product are:

1. define a simple structure to carry the calibration data in a Product

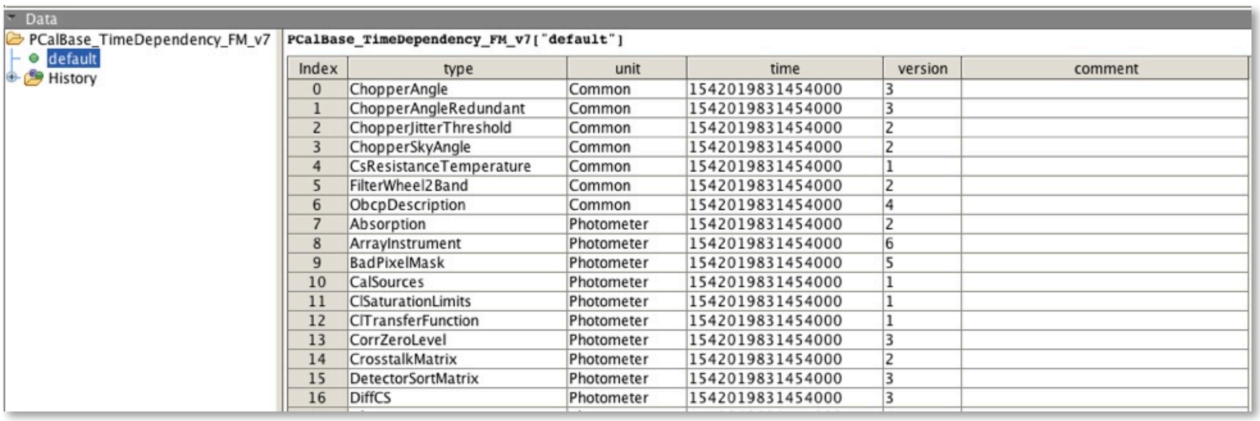
2. create and populate such a Product
3. define access methods to the calibration data
4. develop a Java class for the calibration product
5. define transformations, conversions for the calibration data
6. develop a support class for the calibration product
7. write a JUnit test for the calibration product and its supporting class

TBW more elaborate information is needed here including examples etc.

3.7. Time Dependency

The time dependency product is a simple look-up table captured in a TableDataset. The table contains five columns: type, unit, time, version, and comment. The *type* is the (class) name of the calibration product, *unit* is the name of the branch, i.e. Common, Photometer or Spectrometer, the *time* column specifies the start time of the validity period, the *version* column is the version number of the calibration product for which this period applies, and the *comment* is a free text entry.

Each row in the table represents a validity period for a specific version of a calibration product. The validity period starts with the time given in the table and ends when a new validity period for another version of that same calibration product starts.



PCalBase_TimeDependency_FM_v7["default"]

Index	type	unit	time	version	comment
0	ChopperAngle	Common	1542019831454000	3	
1	ChopperAngleRedundant	Common	1542019831454000	3	
2	ChopperJitterThreshold	Common	1542019831454000	2	
3	ChopperSkyAngle	Common	1542019831454000	2	
4	CsResistanceTemperature	Common	1542019831454000	1	
5	FilterWheel2Band	Common	1542019831454000	2	
6	ObcpDescription	Common	1542019831454000	4	
7	Absorption	Photometer	1542019831454000	2	
8	ArrayInstrument	Photometer	1542019831454000	6	
9	BadPixelMask	Photometer	1542019831454000	5	
10	CalSources	Photometer	1542019831454000	1	
11	ClSaturationLimits	Photometer	1542019831454000	1	
12	ClTransferFunction	Photometer	1542019831454000	1	
13	CorrZeroLevel	Photometer	1542019831454000	3	
14	CrosstalkMatrix	Photometer	1542019831454000	2	
15	DetectorSortMatrix	Photometer	1542019831454000	3	
16	DiffCS	Photometer	1542019831454000	3	

The time is given as a long representation of FineTime[※]. Since the lookup algorithm sorts the filtered table on time, there is no need to keep the table sorted in a particular order (other than easier maintenance and visual inspection).

There is a separate time dependency table for each instrument model and scope.The time dependency tables are treated in the same way as the calibration products. That means they must have the same meta data and filename convention as any other PACS calibration product. The only difference for the file name is that the unit/branch is replaced by the scope of the time dependency information, i.e. BASE, TEST or BULK.

The version of the time dependency table is the version of the calibration tree, i.e. the calTree itself has no notion of versions, it is the time dependency product that determines the version.

For design purposes the time dependency product is attached to each branch in the calTree instead of to the calTree itself. When inspecting a branch, you will therefore see `timedep` as one of the products. It is however currently not possible to dot-navigate to this `timedep` product.

3.8. Configuring the Calibration Framework

3.8.1. Properties

In this section, we explain which properties are understood by the framework. It's important to remember that it is unnecessary to set any of these properties if you are using the default settings that are defined by the framework. In fact, it is better not to specify these properties when using the defaults, because then the framework has the liberty to change names and location and your installation will pick this up automatically. When you set any of these properties yourself, framework changes will not be picked up by your installation.

hcss.pacs.cal.store.type

The calibration framework understands different kinds of storage for calibration products. By default, the calibration products are distributed as a directory of FITS files. This property is therefore set to 'directory' in the distribution. If you use a local store to access you calibration products, set the property to 'lstore'. By default, this property is set to 'directory'.

```
hcss.pacs.cal.store.type = directory
```

hcss.pacs.cal.store.id

This property specifies the identifier for the pool or storage to be used by the framework. This pool/storage is supposed to provide access to the PACS Calibration Products. The property is used by the implementing classes of the CalStoreFactory interface. This property is not used when the type is 'directory', with 'lstore' the default setting is:

```
hcss.pacs.cal.store.id = cal_3405691582
```

hcss.pacs.cal.store.dir

This property specifies the location of the store holding the calibration products. The value of this property is dependent on the type of storage. For a 'directory', this property points to the location of the FITS files, for an 'lstore' this property points to the location of the local store. i.e. where the pool with 'id' is located. Please note that for the 'directory' type, this property is now replaced by a preference which takes precedence over the property. This property is only still used by 'directory' as a default for the preference.

The latest calibration store is always delivered with the hcss.dp.pacs distribution and is located in the directory denoted by:

```
hcss.pacs.cal.store.dir = ${var.hcss.dir}/data/pcal
```

If you want to access the calibration store from your session for inspection, the easiest way is to get hold of the storage where the calibration products are stored

```
calStorage = getCalStorage()
```

Then use the *Product Browser* View in HIPE to access this calStorage.

hcss.pacs.cal.store.factory

This property contains the fully qualified name of the class to be used for creating the storage that will be used to access the calibration products. That class must implement the CalStoreFactory※ interface which has only one method, i.e. getStorage() returning a ProductStorage.

The framework provides several implementations for this interface. Most of these implementations make use of the properties described above, but not all. Consult the Javadoc[3] for these factory classes for specific information on the properties that are used and the product storages that are returned.

The default setting for this property is

```
hcss.pacs.cal.store.factory = herschel.pacs.cal.util.DefaultCalStoreFactory
```

3.8.2. Switching configuration in HIPE

You do not need to go out of HIPE in order to switch your calibration framework setup. The factory manager can be re-initialized and will then read the changes in properties that you set using Configuration. The factory manager is not imported by default, so it needs to be imported first, then make your property changes, and then initialize the factory manager. After that, the calibration tree needs to be read again and you can continue working with your new setup.

The example below sets the calibration framework to use the calibration files for the flight spare located in my home directory `~/work/pcal_flight_spare`.

[3] The PACS Developer Reference Manual, API documentation, in the HIPE on-line help. Go to herschel.pacs.cal.util.CalStoreFactory to see which classes implement this interface.

```
from herschel.pacs.cal.util import CalStoreFactoryManager

Configuration.setProperty("hcss.pacs.cal.store.factory", "herschel.pacs.cal.util.DefaultCalStoreFactory")
Configuration.setProperty("hcss.pacs.cal.store.type", "directory")
Configuration.setProperty("hcss.pacs.cal.store.dir", "${user.home}/work/pcal_flight_spare")

CalStoreFactoryManager.initialize()
calTree = getCalTree()
```

3.9. Framework specifics for Pipeline Processing

TBW Which factories to use, calibration products from the HSA, no queries allowed during processing, calibration plugin.

3.9.1. Calibration Plugins

The Pipeline processor uses a plugin mechanism to initialize the calibration tree that needs to be used for pipeline processing. These plugins are implemented in the `pacs_spg_plugins` package, which is not part of the calibration framework, but for completeness we will describe the plugins here anyway.

The operational baseline

The plugin that is used by the operational baseline is the PacsCalPlugin. This plugin initiates a PACS calibration tree for the Flight Model FM and scope BASE. The start time of the observation is used for time dependent products.

```
calibration = PacsCal.newInstance(PacsConfig.FM, PacsConfig.BASE_SCOPE, obs.getStartDate())
```

The plugin additionally checks if the property `hcss.pacs.spg.plugins.base.version` is set. This property is of type integer and specifies the version of the calibration tree. The property can be used to fix the calibration tree to a specific version.

Bulk re-processing

The plugin that is used for bulk re-processing is the PacsCalBulkPlugin. This plugin initiates a fixed version of the PACS calibration tree for the Flight Model FM and scope BULK. The start time of the observation is used for time dependent products.

```
calibration = PacsCal.newInstance(PacsConfig.FM, PacsConfig.BULK_SCOPE, obs.getStartDate(),
                  version)
```

The plugin needs the property `hcss.pacs.spg.plugins.bulk.version` to be set. This property is of type integer and specifies the version of the calibration tree. The property must be used to fix the calibration tree to a specific version. If the property is undefined, an exception will be thrown.

3.10.Utility functions provided by the framework

3.10.1.Running the Calibration File Checker

Any change to a calibration product must be verified by the CalFileChecker. Just run the utility from the unix command line specifying the cvs checkout directory as an argument.

```
$ calfilechecker ~/workspace/pacs_data_pcal/develop/data/pacs/data/pcal
```

This will report any inconsistencies in the meta data and the file name for each calibration product in the target directory. Run this command each time before committing updates to cvs.

4. The Developer Manual

4.1. Introduction

TBW

5. The Framework Developer Manual

5.1. Caveats

We have a small design problem for calibration products that extend Context because we start from a directory containing FITS files for all calibration products, and we currently do not know how to handle the Contexts in this directory. Until now there are no calibration products defined as Contexts.

5.2. TBW

- The use of Preferences in test harnesses. See HCSS-11921 and HCSS-11900.
- Description of the GUIs (layout and the names of the Panes).
- Authenticator
- Deprecation of importFrom methods for CalibrationProduct
- How to implement a new calibration product?

5.3. Introduction

TBW

TimeDependency version <-> calTree version…

TimeDependency look-up table: The algorithm to determine the cal product version for a given time first filters the table for the calibration product and unit, then sorts the filtered table by the time column, and then selects the version for which the largest time is less or equal to the specified time.

5.4. Time Dependency

The time dependency of calibration products is defined in a simple table which associates the versions of calibration products with a valid time period. This table is implemented as a TimeDependencyTable class which extends a TableDataset. This TimeDependencyTable is wrapped in a TimeDependency product which extends a AbstractCalibrationProduct. A time dependency table is therefore also a calibration product and treated as such.

The TimeDependency calibration product is be attached dynamically to each branch in the calibration tree, at the time the branch is requested from the tree, because at that time the branch will be disconnected from the tree and should be self contained, i.e. it must know its applicability time and the corresponding time dependency.

5.5. Which classes are dynamically created?

Which of the classes are dynamically created during your session, and which are loaded from a pool or another external location? The calibration products are obviously loaded from either a pool or the local file system. The PacsCal class is dynamically created only in your session. The calibration tree (PacsCal) can not be loaded from a pool because the tree is dynamically created based on a time that is passed into the constructor either directly or via an observation context.

Appendix A: User Commands

The getCalTree() command

Description

This HIPE command loads a calibration tree into your session.

Synopsis

```
calTree = getCalTree(model="FM|FLIGHT|FS|QM", scope="BASE|TEST|BULK", time=time, obs=obs,
                     version=version, verbose=True|False)
```

Arguments

All arguments to getCalTree() are optional. If no arguments are specified, the command loads a calibration tree with the latests applicable calibration products for model="FM" and scope = "BASE".

model

The instrument model for which you need the calibration tree, i.e. one of "FM", "FLIGHT", "FS", or "QM". The type of this parameter is String and the default value is "FM". The instrument model is explained in more detail in the section on instrument models in the user manual chapter.

scope

The scope for which a calibration tree is requested, i.e. one of "BASE", "TEST", or "BULK". The type of this parameter is String and the default value is "BASE". The scope is explained in more detail in the section different scopes for calibration trees in the user manual chapter.

time

The time for which the time dependent calibration products in the calTree will be selected. The time must be a FineTime or a String in the format: "dd-MMM-yyyy HH:mm:ss". This argument has precedence over the `obs` argument.

obs

An observation context for which the calibration tree should be selected. The start date of the observation is picked up in order to select the correct time dependent calibration products. This argument is ignored if a time argument is given.

version

The version of the calTree to be loaded. The version is explained in more detail in the section versions of calibration trees in the user manual chapter. By default, when no version is specified, the latest version is loaded.

verbose

If set to True, the command will print information about the location of the calibration products on the console. The default value is False.

Examples

```
calTree = getCalTree()
calTree = getCalTree("FM", "BASE")
calTree = getCalTree("FM", "BULK", version=6)
calTree = getCalTree(model="QM", scope="TEST", time="17-Oct-2004 12:09:00")
calTree = getCalTree(model="QM", scope="TEST", obs=myObservation)
```

The getCalProduct() command

Description

This HIPE command loads a calibration product into your session.

Synopsis

```
cal = getCalProduct(unit, name, version=-1|version, model="FM|FS|QM", scope="BASE|TEST|BULK",
                    verbose=False|True)
```

Arguments

The first two arguments are mandatory and specify which calibration product shall be loaded. The other arguments are optional.

unit

This is the name of the instrument unit for which the calibration product is requested. Accepted values for unit are "spectrometer", "photometer", and "common".

name

The name of the calibration product. You can find the valid name by printing the calTree from within HIPE, e.g.

```
HIPE> calTree = getCalTree()
HIPE> print calTree.common
PacsCalCommon Calibration Products:
  chopperAngle                    : FM, 3
  chopperAngleRedundant           : FM, 3
  chopperJitterThreshold          : FM, 2
  ...
```

version

The version of the calibration product to be loaded. By default, when no version is specified, the latest version for the given scope is loaded. The type of this parameter is integer. *Beware that this is not necessarily the last version available in the calibration pool*.

model

The instrument model for which you need the calibration product, i.e. one of "FM", "FS", or "QM". The type of this parameter is String and the default value is "FM". The instrument model is explained in more detail in the section on instrument models in the user manual chapter.

scope

The scope for which a calibration product is requested, i.e. one of "BASE", "TEST", or "BULK". The type of this parameter is String and the default value is "BASE". The scope is explained in more detail in the section different scopes for calibration trees in the user manual chapter.

verbose

If set to True, the command will print information about the location of the calibration products on the console. The default value is False.

Examples

```
ai = getCalProduct("Spectrometer", "ArrayInstrument", 3)
ai = getCalProduct("Spectrometer", "ArrayInstrument", scope="BULK")
ca = getCalProduct("Common", "ChopperAngle", 2, "QM")
```

Glossary

branch

CalStoreFactory

This is an interface ….

calTree

Short hand for Calibration Tree. This is explained in detail in the first chapter, the user manual.

CamelCase

a notation where compound words are joined and their first letter is capitalized, e.g. DetectorSelectionTable. See the Wikipedia entry on CamelCase.

cvs

Concurrent Versioning System...

dot-navigate

I use this to indicate that you can access a product or data within a product by concatenating names with dots until you reach the desired information.

```
ftr = calTree.photometer.filterTransmission.red
```

ESA

European Space Agency

FineTime

Time representation used in the HCSS.

Framework

A framework is a collection of classes, tools, and APIs to help the different components in a system to work together. Find a good definition of a software framework at Wikipedia.

HCSS

Herschel Common Software System. Where to download?

HIPE

The Herschel Interactive Processing Environment is the public software package that is a joint development of ESA, NHSC and the HIFI, PACS and SPIRE instrument consortia. HIPE is part of the HCSS and is licensed under the GNU Lesser Public License.

HSA

Herschel Science Archive. It is the primary source of data for the astronomical community of Herschel. The archive is maintained by ESA at their science centre in Spain (ESAC). In order to access the archive you will need a username and password. More information can be found at ESA Herschel Science Archive Website.

HSC

The Herschel Science Center is responsible for running the science operations of the Herschel satellite.

ICC

The Instrument Control Center is responsible for monitoring the health of their instrument and for the in-orbit calibration of the instrument. For this specific engineering and calibration observation are scheduled at regular intervals.

JIRA

JIRA is the software problem report system that is used by the HCSS development team and the ICCs. JIRA is also used to track deliveries of updated calibration products from the ICCs to the HSC.

NHCS

NASA Herschel Science Center

TableDataset

This is a data structure used in the HCSS to represent a table. Which Viewer? Other important information about TableDataset?

unit

This name is used interchangeably with branch, however branch is used in connection with the structure of the calTree, whereas unit is used when referring to the instrument.