SUBJECT: **Calibration Product Framework**

PREPARED BY: **S.Guest**

DOCUMENT No: **SPIRE-RAL-NOT-002240**

ISSUE: **1.0** **Date:** **13th January 2005**

Change Record

ISSUE	DATE	
0.1 *draft*	11th October 2004	First draft
0.1 *draft 2*	1st December 2004	Second draft
1.0	13th January 2005	First wider distribution

Table of Contents

1 INTRODUCTION

1.1 Acronyms

API	Application Programmer Interface
FITS	Flexible Image Transport System
HCSS	Herschel Common Science System
IA	Interactive Analysis
ISO	Infrared Space Observatory
SPG	Standard Product Generation
TBD	To Be Defined

1.2 Conventions used in this document

The `courier` font is used to denote a literal string, such as a package name within HCSS or IA, for example `herschel.ia.dataset`.

Italics are used to indicate that a word or phrase has a specific technical meaning in that context, for example *association* in a database context.

1.3 Purpose

The main purpose of this document is to specify baseline requirements for a framework to support calibration products within a HCSS/IA environment. A secondary purpose is to record some preliminary design considerations with regard to implementing these requirements. It is therefore intended as a single point of reference for the development of this framework.

2 PRINCIPLES AND REQUIREMENTS

2.1 Product format

Calibration products are normal IA products and are created by means of the `herschel.ia.dataset` package[1]. This ensures a wealth of already available functionality such as import/export with FITS files and the database. Moreover it enables users to design their own calibration products. The scope of these products is not limited to those used in Standard Product Generation (SPG). Calibration products in addition to those used by the pipeline can be delivered to the observer in order to allow further interpretation of the data.

As a goal, each calibration product should contain a full *history* describing how it was created. However, this should not be taken to imply a restriction on what can be a calibration product. The requirements on exactly what constitutes the history are unclear at this time and possibly outside the scope of this document. It should also be possible to create a textual note in the product describing its contents.

Metadata names should follow the normal conventions as defined by the `herschel.ia.io` package. Names already defined should be reused (i.e. the mappings to FITS keywords for FITS standard keywords, HEASARC FITS conventions and HCSS extensions). More standard names should be agreed so that the same things have the same names across the Herschel project.

Products should contain a version number as a metadata item, since a time stamp, while necessary, is not sufficient. Version identifiers should be assigned automatically – the user should not have to issue a database query to find what the last one was and hard-code the next value in a product generation script. Note that the general IA requirement of not being coupled to Versant still applies, see design considerations for some thoughts on how this could be reconciled. Products should be automatically time-stamped on creation. Both the time stamp and the version identifier should be human-readable.

There is an assumption here that each *version* of a product is stored as a separate product, rather than a single product containing multiple versions. Note that this does not preclude storing all values of time-dependent parameters within a single product. In this case the product would grow in size over the course of the mission i.e. each *version* would be larger than the last.

2.2 Data Access

It must not be necessary to have a database licence in order to access calibration products. It must also be possible to retrieve the products through a firewall.

It should be possible to get a calibration product based on:
1. The *type* of the product.
2. The latest version
3. A specific version (by version identifier or time)
4. Applicable to given time range
5. Applicable to given instrument model
6. Applicable to certain observing and instrument modes
7. Applicable to certain criteria e.g. temperatures

[1] This is an unusual requirement in that it specifies an implementation. Nevertheless, at this point in Herschel development it is a genuine user requirement.

8. An exact match of arbitrary metadata items
9. A "best fit" algorithm most closely matching arbitrary metadata items
10. A "plug-in" algorithm to allow searching of data as well as metadata
11. Combinations of the above

It should be possible to locate products both by direct *query* on the products, and by *navigation* from *associated* objects such as observations, observing modes etc. Users will require a *browser* (or browsers) to perform these functions (see “Tools” section), while developers will require a well-defined API.

It is perfectly valid for a query to return a result set with a size greater than one. However, for pipeline purposes it is necessary to be able to guarantee a result set with size either zero (not found) or one (found). The “best fit” algorithm referred to above can be used for this purpose.

At least two options should be provided to control the logic of what to do when some versions of a product contain metadata items specified in a query and some do not:

1. The metadata item is ignored for filtering purposes i.e. that product is still matched. This allows additional metadata items to be added to later versions of a product, while still being able to find earlier versions with the same queries. However, for “best fit” purposes, a match must be preferred to a non-match and assigned a higher weight.
2. No versions of the product that do not contain the item are matched. This enables old versions containing insufficient information to be deliberately excluded.

This behaviour should be controllable by a switch.

It should also be possible to add metadata items to previous versions if it is subsequently found that these items are needed to be searchable.

2.3 Data Organisation

This section describes calibration product organisation in database terms. It should be possible to *export* these data structures into a file system with a 1-1 product-file mapping.

There is an identified need for a more complex hierarchy than a "flat" product set (plus versioning) i.e. a single “directory” is not enough. For example, products could be organised in terms of photometer, spectrometer, subsystem, general etc. It should be possible to create any hierarchy in a similar manner to a directory-based file system. It should also be possible to create multiple disconnected hierarchies (or trees). Note that a single flat structure is also perfectly valid.

It should be possible to *associate* these hierarchies with root objects in the database. The full set of these root objects is not currently well-defined, but are expected to include:

- Observations. This provides a link between standard products from SPG and the calibration products used to derive them. The converse link is also required in some cases such as trend products, where it is necessary to determine which observations were used to create the calibration product.
- Instrument Model.
- A *configuration* of an *instrument model* of an *instrument*, where a *configuration* consists of a set of *configuration items*. This is a more complex elaboration of the previous bullet. Examples of configuration items include hardware parts e.g. detector array, on-board software, a default instrument setting e.g. bias, AOT logic etc. In this scenario a calibration product describes the behaviour of one or more configuration items.

- Observing mode. Note that an observing mode from a user point-of-view is not necessarily the same thing as a mode from an instrument point-of-view.

The difficulty in formulating precise requirements on these associations implies a requirement for flexibility.

2.4 Processing

Standard products as produced by the pipeline must contain in their history a description of which versions of calibration products were used in their creation. It is equally important to record which versions of tasks were used in creation of a product – this applies to both calibration and non-calibration products. These are the minimum requirements on history. It should be possible to easily retrieve the calibration products that were used to create a product.

The equivalent of the ISO CAL-A and CAL-B pipeline products can be seen as intermediate processing products of the SPG rather than as calibration products.

Interpolation between data points should be supported for the cases where values vary over time but are only measured at discrete intervals. This interpolation applies between rows of a table, and possibly (TBD) also between datasets of the same shape and dimension within a product. It should be possible to "plug-in" the interpolation algorithm used.

2.5 Tools

The following tools are needed to support the users in creating calibration products:

- A dataset editor to be able to modify and create datasets. This could be an extension of the existing Dataset Inspector.
- A product hierarchy browser, similar to a file system browser. At least the description and metadata should be displayed. It should be possible to see an overview of the different types, and to be able to set which version of a product is the one that will be used by default by IA and SPG.
- A simple means of comparing versions, at least in terms of seeing differences in metadata and data size.

2.6 Use Cases

The following Use Cases are intended as clarifications of some of the points already mentioned.

2.6.1 Create a calibration product

1 USR: Use IA to create a product.
2 USR: Specify the *type* of the product.
3 USR: Save the code to create the product as a script.
4 IA: Save the script as a file

Extensions:
1a USR: create a script in favourite editor
1b USR: load script into IA

2.6.2 Save a calibration product

1 USR: Request product to be saved in the database.
2 IA: Examines the metadata to determine where the product should be placed and which associations should be made.

3 IA: Create a new version of the product and assign a version identifier.

Extensions:
1a USR: Specifies the location where the product should be saved.
1b USR: Requests the product be saved as a local file e.g. in FITS format.
2a IA: Give a message indicating that the metadata is insufficient to determine what to do with the product.
3a IA: Overwrite a previous version (if requested as an option).
3b IA: Don't create a new version if metadata is identical to a previous version and give an error message (if requested as an option).
3c IA: Create an entirely new product, not a new version (if requested as an option).

2.6.3 Retrieve a calibration product

1 USR: Specify criteria for product(s) to retrieve. See section on Data Access requirements.
2 IA: Return an iterator over all matching products.

Extensions:
1a SPG: Specify criteria for product to retrieve.
2a SPG: Return a single "best match" product.

2.6.4 Browse calibration products

1 USR: Start browser tool
2 IA: Display starting level information
3 USR: Refine the browse search
4 IA: Display the refined information

Extensions:
3a USR: Change the default *version* of a product.
3b1 USR: Request difference between two versions of a product
3b2 IA: Display differences

2.6.5 Create a product node in hierarchy

1 USR: Specify location in database to create a new node.
2 IA: Check that the user is permitted to create a node at the requested location.
3 IA: Create the new node.

Extensions:
1a USR: Specify a node in which to create a subnode.
1b USR: Specify which set of *rules* will apply to this node.

2.6.6 Retrieve calibration products used to generate a pipeline product

1 USR: Specify a pipeline product
2 IA: Retrieve the versions of calibration products that were used to generate the product.

Extensions:
2a IA: Retrieve the latest versions of the same calibration products as were used to generate the product.

3 NOTES AND PRELIMINARY DESIGN CONSIDERATIONS

3.1 Product Format

The version number could be automatically added when the product is stored in the database. However this then only works with the database.

It is possible that the "textual note" implies a more elaborate description than is intended by the *description* field of a product.

It is unclear whether or not a calibration product should be a *subclass* of a *product*. If so, a related question is how complex do we make the calibration products? For example: if a calibration product class holds the polynomial coefficients of a polynome does it know it is a polynome? Does it return a dataset containing the coefficients to a task using it, the coefficients themselves, or a Polynome object? Allowing calibration classes to become too complex might contradict the requirement for users to be able to define their own types.

The dataset package allows for products to be defined which are functions. For example, the frequency calibration in a spectrum dataset can be a table (column) and a function (defined in the metadata field). This feature has some advantages (i.e. clean uncertainty estimates in the derived frequency scale) and (perhaps) some disadvantages. For example a possible disadvantage might be that an actual frequency scale is not stored, but the means for calculating one is. In this case a frequency scale is not directly exportable. There does not appear to be any problem with this scheme being applied to calibration products – searches are generally metadata based and those that inspect the data itself do so by means of a "plug-in" algorithm that can apply the function to the data.

3.2 Data Access

Some of these requirements are already supported by the `herschel.access` and `herschel.ia.io.dbase` packages.

The requirement to be able to add metadata items to earlier versions has certain implications. Possibilities are:

1. The products are not considered to be immutable and can be replaced. This possibility is supported by the product node code. It is probably the simplest.
2. Sub-branching could be introduced e.g. version 2 with added metadata could become 2.1. This is not currently supported by the product nodes.
3. The sub-branching above could be implemented as the current versioning supported by the product nodes, and another layer would be needed to support the main branch versions.

3.3 Data Organisation

"Product nodes", as defined in the `herschel.ccm.api.product.node` package, should be used to store products in the database. It also supports the setting of a set of *rules* to determine which operations are permitted on a node. This software infrastructure already exists. See the package documentation for full details. The `herschel.ia.io.dbase` also contains a "save and retrieve" API on top of this functionality.

There are some open issues with regard to how associations are represented in the database. For example:

- Should each observation be associated with the set of all its applicable calibration products? Note that in the database it would only be a pointer being duplicated, not the data (in the file system analogy, this corresponds to a link or shortcut).
- Are all the versions of a calibration product associated with an observation, or just the versions that are applicable to it? The latter is not really consistent with the "product node" design.

The various data processing scenarios should be considered in answering these questions as the needs of the typical IA user can be quite different from the needs of the data processing pipeline.