Max Planck Institute for extraterrestrial Physics

Recording the History of Herschel* Data Processing in Products

Jeroen de Jong[1](jdejong@mpe.mpg.de), Rik Huygen[2], Jorgo Bakker[3], Ekkehard Wieprecht[1], Roland Vavrek[3], Michael Wetzstein[1], Juergen Schreiber[4], Eckhard Sturm[1], Stephan Ott[3]

[1]Max Planck Institute for Extra-Terrestial Physics, Garching bei München, Germany; [2]Institute of Astronomy, KU Leuven, Belgium; [3]ESA Herschel Science Centre, Madrid, Spain; [4]Max Planck Institute for Astronomy, Heidelberg, Germany

Overview

For the scientific assessment of any products generated by a Data Processing (DP) system it is important to know how the product has been generated. Therefore, such a DP system should record the history of tasks/recipes used to generate the product.

We present how the history is recorded during data processing with the Herschel[2] Common Software System (HCSS[1]**), and how users can inspect this in the Herschel Interactive Processing Environment (HIPE[1]). The Herschel DP software records after the execution of each (pipeline) task all information needed to redo the task. This includes:

- the name of the task
- the build number
- the used input parameter names and their values
- in case an input parameter is a product: the complete history of that product and if available a human readable identifier (such as a calibration filename).

In this way one creates in the final product the complete chain of tasks needed to reproduce that product from the raw data. The user can inspect this chain with a convenient viewer in HIPE and even retrieve it as a runnable Python script. This script can one rerun with different parameters and/or e.g. calibration files.

When the product is saved in a FITS file then all the history information is stored in two binary FITS tables (one for the tasks and one for the parameters). Additionally the corresponding Python script is saved in an additional table, just for convenience in case users want to inspect the file with other software.

How is the history stored in the products?

The history is stored in three tables:

- **HistoryTasks:** Contains the task ID, name, execution date and used software build number.
- **HistoryParameters:** Contains all the information about the parameters used in the task. The rows in this table are linked to a task by the task ID.
- **HistoryScript:** Contains just the generated python script for reference. Provided for users who quickly want to check the history outside of HIPE.

These tables are stored in FITS files in 3 binary FITS table HDUs. A detailed description of the table columns and their links is shown below.

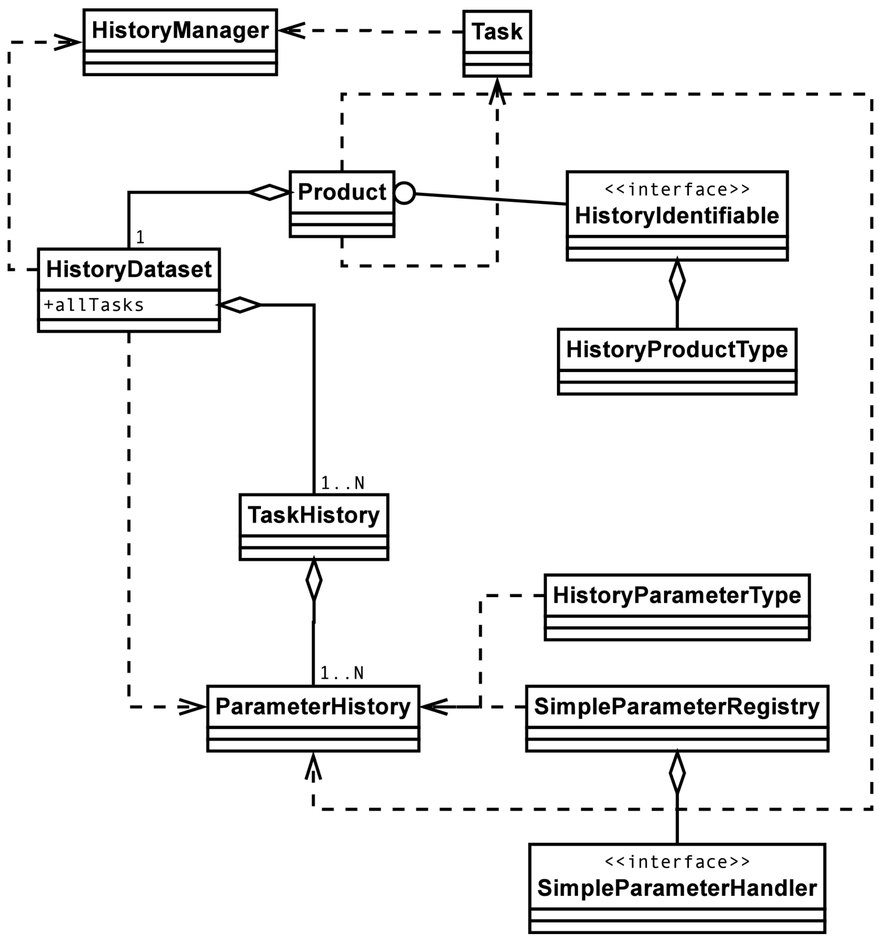
HistoryTasks Table

Column	Format	Description
ID	Long	Unique ID which links the tasks and parameters
Name	String	Name of the task
ExecDate	Long	Timestamp of execution
BuildVersion	String	The software build which was used (e.g. 4.0.1130)

HistoryParameters Table

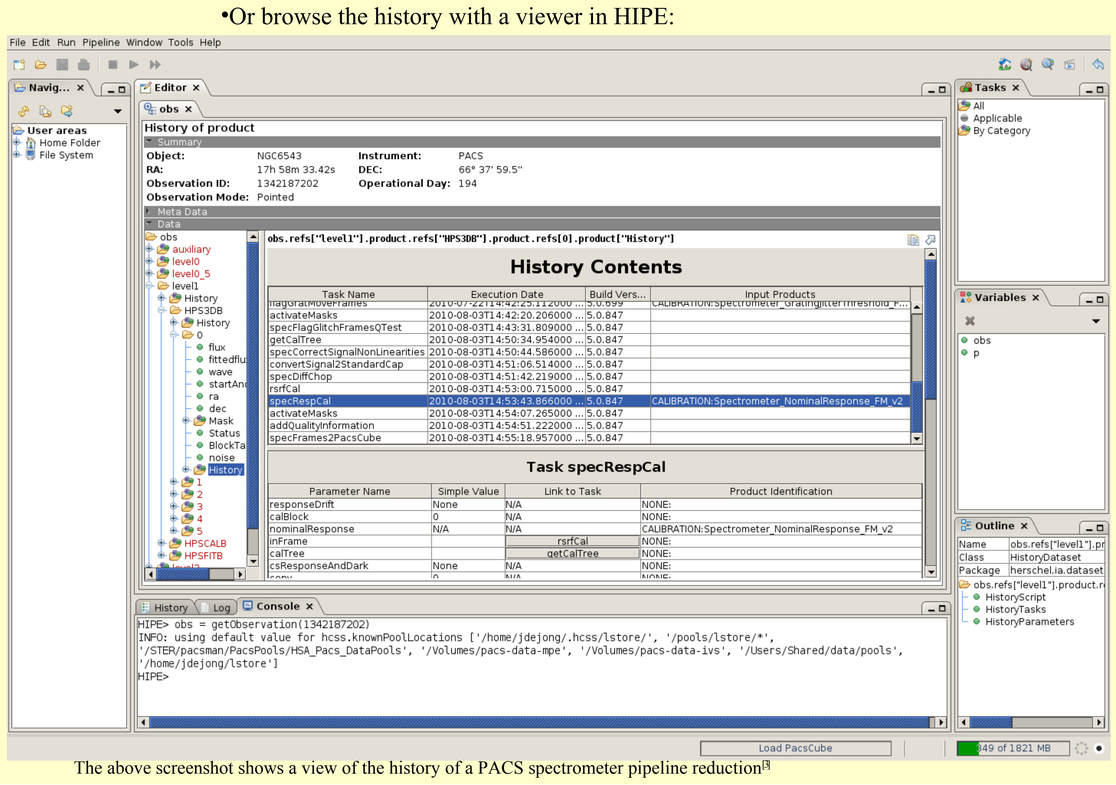
Column	Type	Description
TaskID	Long	ID which links the parameter to its task
Name	String	Name of the parameter
Type	String	Type of parameter (simple STRING, INTEGER, etc., PRODUCT, OBJECT (generic))
Value	String	The simple value of the parameter (when available)
IsDefault	Boolean	Was the default parameter value used?
IncTaskId	Long	If applicable: The ID of the task which generated this product. Used to merge the history of products.
UserInput	Boolean	Needs user input? (neither a product nor a simple value)
Class	String	The Java class of the value object
ProductType	String	Type of Product (e.g. CALIBRATION)
ProductId	String	A human readable identification (when available)

Class Diagram



How users can examine the history

- One can simply print the history on the console in HIPE:
  `print someProduct.history`
- Or save it as a Python script which could be edited:
  `someProduct.history.saveScript("script.py")`
- Or browse the history with a viewer in HIPE:



The above screenshot shows a view of the history of a PACS spectrometer pipeline reduction[3]

Important APIs

HistoryManager - Main interface between tasks and history
addTask(Task t) - Add a task to the history

HistoryDataset - Contains the history tables
getScript() - Returns the history as Python script
saveScript(String file) - Saves the Python script in a file
getAllTasks() - Gets all the tasks in the history
toString() - Produces a human readable overview

TaskHistory - Contains the history of a single task
getName() - Gets the task name
getExecDate() - Gets the execution date
getVersion() - Gets the build number
getParameters() - Gets the list of parameter names
getParameter(String name) - Gets a single parameter

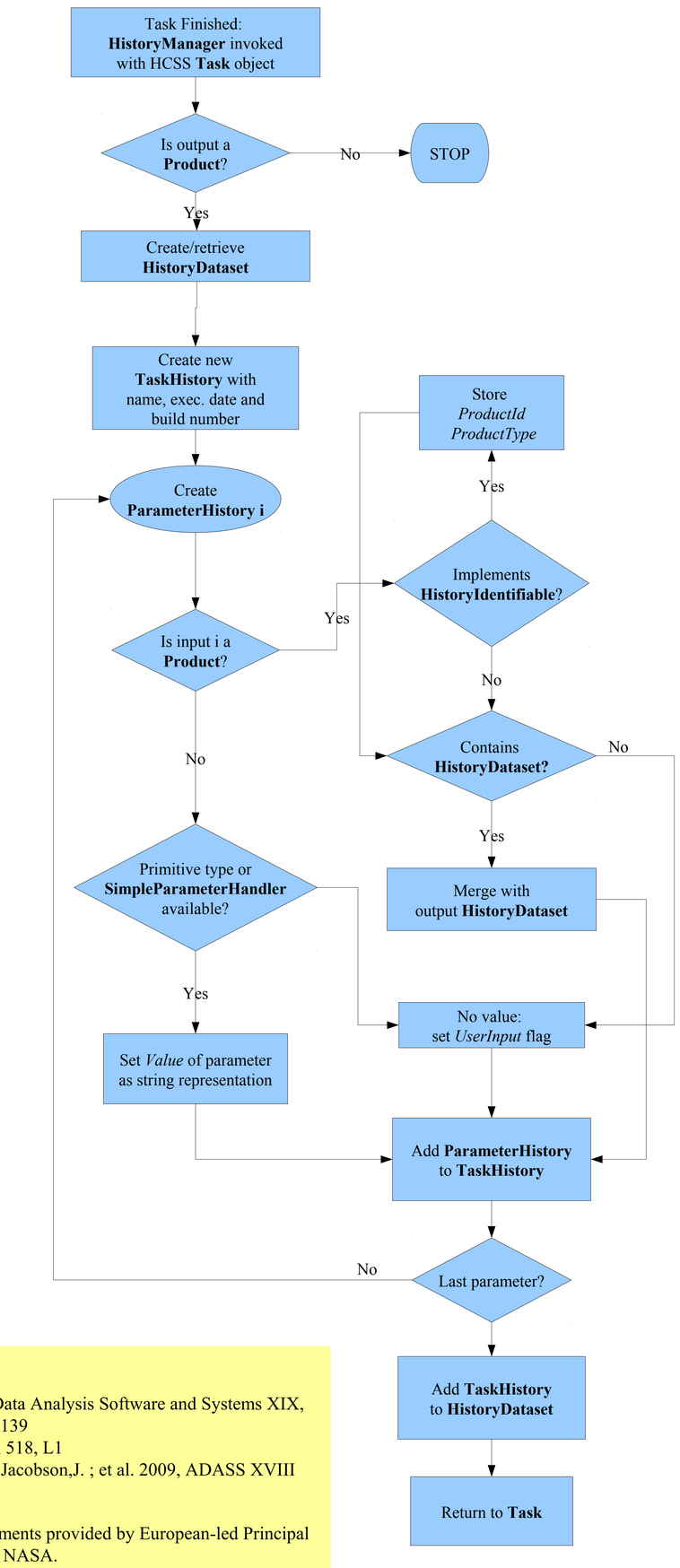
ParameterHistory - Contains the history of a single parameter
getName() - Gets parameter name
getType() - Gets type of parameter
getValueAsString() - Gets the parameter value
getProductIdentifier() - Gets Product ID (when available)
getProductType() - Gets Product Type (when available)
getValueClass() - Gets the class of the parameter value
isDefault() - Has the parameter its default value?
isUserInputNeeded() - Does the user need to provide its value?

HistoryIdentifiable - Interface which produces a short human readable product identifier for the history (used in ParameterHistory)
getHistoryProductType() - Returns type of product
getHistoryIdentifier() - Returns a human readable short ID of a product

SimpleParameterHandler - Interface which defines how to convert an arbitrary object value to/from a string value for the history
canHandle(Class cls) - Determines whether this handler can handle the given class.
canHandle(Object obj) - Same for an object
getSimpleValue(Object value) - Converts an object value to a string
getValue(String simpleValue, Class cls) - Creates an object from a simple string value.
getPythonRepresentation(String simpleValue, Class cls) - Gets the Python code to construct the value from a simple string value in a script.

SimpleParameterRegistry -
A registry for SimpleParameterHandler implementations
getInstance() - Gets the instance of the registry
register(Class cls, SimpleParameterHandler handler) - Registers a handler

Flow diagram of history recording



References

[1] Ott, S. 2010, in ASP Conference Series, Astronomical Data Analysis Software and Systems XIX, Y. Mizumoto, K.-I. Morita, and M. Ohishi, eds., vol 434, p. 139
[2] Pilbratt, G.~L., et al. 2010, Astronomy & Astrophysics, 518, L1
[3] Schreiber,J. ; Wieprecht,E. ; de Jong,J. ; Wetzstein,M. ; Jacobson,J. ; et al. 2009, ADASS XVIII ASP Conference Series, Vol. 411, p.478

* Herschel is an ESA space observatory with science instruments provided by European-led Principal Investigator consortia and with important participation from NASA.

** The Herschel Common Software System (HCSS) is implemented using JAVA technology and written in a common effort by the HERSCHEL Science Center and the three instrument teams.