Tessella Project Number: 7596

**Tessella plc** 26 The Quadrant, Abingdon Science Park, Abingdon, Oxfordshire, OX14 3YS, UK

T: +44 (0)1235 555511 | F: +44 (0)1235 553301 | E: info@tessella.com | www.tessella.com

Client: RAL & ORNL

Mantid

Nested History Detailed Design Document



Document Control

This document is under document control. All members of the distribution list will receive updated copies whenever alterations are made. All other copies are uncontrolled.

Document Owner

|  |  |  |
| --- | --- | --- |
| Name | Role | Company |
| Nicholas Draper | Owner | Tessella |

Document Approval

The following are required to approve this document. Their approval signifies that the document is fit for purpose and approved for use. .

|  |  |  |
| --- | --- | --- |
| Name | Approval Role | Company |
| Martyn Gigg | Subject Expert | RAL |
| Michael Reuter | Subject Expert | ORNL |

Distribution

|  |  |  |
| --- | --- | --- |
| Name | Role | Company |
| Mantid GIT repository | Filing | GitHub |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Author | Issue | Date | Description |
| Nicholas Draper | V1.R1.M0 | 03-Jan--2013 | 1st version written |

References

|  |  |  |  |
| --- | --- | --- | --- |
| Document | Ref | Date | Details & Issue |
| Architectural Design Document | [ADD] | 11-Nov-2009 | V1.R2.M1 |

Table of Contents

[1 Introduction 3](#_Toc344987998)

[1.1 Definition of Terms 3](#_Toc344987999)

[2 Problem: 4](#_Toc344988000)

[3 Proposed solution 4](#_Toc344988001)

[3.1 Refactoring the history structure to store nested history records 4](#_Toc344988002)

[3.2 Recording the nested history records correctly 6](#_Toc344988003)

[3.3 Displaying and scripting the Nested history records 8](#_Toc344988004)

# Introduction

Purpose of this DocumentThis document describes the detailed design of the Event Workspace component of the Mantid Framework.

It is based on the design specified in the Architectural Design Document [ADD]

It will form the basis of the development of this aspect of the framework and act as a guide for maintaining the system.

Scope of this DocumentThese requirements cover the development of the Event workspace aspect of the Mantid Framework.

Context of this IssueThis is the first draft of the EW-DDD derived from the ADD and after internal review will be updated and used as a basis for the development of the system.

## Definition of Terms

|  |  |
| --- | --- |
| ADD | The Architectural Design Document (this document), the high level design document for the entire system. |
| URD | The User Requirements Document, records the users’ requirements for the system. |
| SRD | The Software Requirements Document, specifies the behaviour of the software system. |
| API | Application Programming Interface, defines the interface through which two programs may interact. |

# Problem:

People want to be able to see what Mantid is doing beneath the surface. Particularly as part of complex workflow algorithms. They would like to be able to script out the workspace history at different levels of depth (or see what a workflow algorithm has done in detail, and be able to reproduce it) in some way.

# Proposed solution

The three main areas that need to be resolved here are as follows:

1. Refactoring the history structure to store nested history records
2. Recording the nested history records correctly
3. Displaying and scripting the Nested history records

These suggested changes clearly affect core functionality and therefore this will have to follow the [Core Functionality procedures](http://www.mantidproject.org/Working_on_Core_Functionality).

## Refactoring the history structure to store nested history records

The current history record structure is a flat list of algorithm history records. This is also used when storing the data into intermediate nexus files, and anything we do will need to be compatible with previous nexus files.

### AlgHistoryIn memory storage

The AlgorithmHistory class will be extended to allow nested records, that is an AlgorithmHistory record can also own within it other AlgorithmHistory records in an internal vector. They should be able to be filled within the constructor, but getters and setters should also be added to allow full access. Within the AlgorithmHistory, only top level parent algorithms will have a valid execCount, all child algorithms will be given an exec count of 0, and will be stored in the order they are added.

**Note**: This should be done by adding overloads and methods to the existing class, and avoid changing existing methods wherever possible.

### Nexus File Storage

The History records are stored within Mantid files, and as such we need to update the loading and saving of this data to support nested records. The overall structure of the records is shown to the right, with one MantidAlgorithm NXnote for each algorithm. In order to preserve the integrity of previous files and keep things simple in the majority of simple cases we will keep this structure, so that there will be one MantidAlgorithm NXnote entry for each top level parent algorithm in the History. Details of Nested History records will be put within the data of the Parent MantidAlgorithm NXnote.

Example of a previous MantidAlgorithm NxNote, and a good example of a new Algorithm with no Child records.

Algorithm: LoadRaw v3

Execution Date: 2009-Oct-09 16:56:54

Execution Duration: 2.3 seconds

Parameters:

Name: Filename, Value: Data/GEM38370.raw, Default?: No, Direction: Input

Name: OutputWorkspace, Value: GEM38370, Default?: No, Direction: Output

Name: SpectrumMin, Value: 1, Default?: Yes, Direction: Input

Name: SpectrumMax, Value: 2147483632, Default?: Yes, Direction: Input

Name: SpectrumList, Value: , Default?: Yes, Direction: Input

Name: Cache, Value: If Slow, Default?: Yes, Direction: Input

Name: LoadLogFiles, Value: 1, Default?: Yes, Direction: Input

Example of new Algorithm with nested Child records. Using separator lines and indentation to improve parsing and readability.

Algorithm: ParentAlg v3

Execution Date: 2009-Oct-09 16:56:54

Execution Duration: 2.3 seconds

Parameters:

Name: Filename, Value: Data/GEM38370.raw, Default?: No, Direction: Input

Name: OutputWorkspace, Value: GEM38370, Default?: No, Direction: Output

Name: SpectrumMin, Value: 1, Default?: Yes, Direction: Input

Name: SpectrumMax, Value: 2147483632, Default?: Yes, Direction: Input Name:

SpectrumList, Value: , Default?: Yes, Direction: Input

Name: Cache, Value: If Slow, Default?: Yes, Direction: Input

Name: LoadLogFiles, Value: 1, Default?: Yes, Direction: Input

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Child Algorithms \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Algorithm: ChildAlg v3

Execution Date: 2009-Oct-09 16:56:54

Execution Duration: 2.3 seconds

Parameters:

Name: InputWorkspace, Value: \_\_Tmp0x234FG2342AB, Default?: No, Direction: Output

Name: OutputWorkspace, Value: \_\_Tmp0x234FG2342AB, Default?: No, Direction: Output

Algorithm: Child&ParentAlg v3

Execution Date: 2009-Oct-09 16:56:54

Execution Duration: 2.3 seconds

Parameters:

Name: InputWorkspace, Value: \_\_Tmp0x234FG2342AB, Default?: No, Direction: Output

Name: OutputWorkspace, Value: \_\_Tmp0x234FG2342AB, Default?: No, Direction: Output

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Child Algorithms \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Algorithm: ChildAlg2 v3

Execution Date: 2009-Oct-09 16:56:54

Execution Duration: 2.3 seconds

Parameters:

Name: Input, Value: \_\_Tmp0x234FG2342AB, Default?: No, Direction: Output

Name: Output, Value: \_\_Tmp0x234FG2342AB, Default?: No, Direction: Output

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Child Algorithms \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Algorithm: DeleteWorkspace v1

Execution Date: 2009-Oct-09 16:56:54

Execution Duration: 2.3 seconds

Parameters:

Name: InputWorkspace, Value: \_\_Tmp0x234FG2342AB, Default?: No, Direction: Output

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Child Algorithms \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Presently the code for writing an AlgorithmHistory class is stored within the AlgorithmHistory.printSelf() method, and will need to be extended to support Child algorithms. The code for Parsing AlgorithmHistory text is in the WorkspaceHistory.loadNexus() method, This will need to be expanded, and refactored such that the parsing of the text itself occurs within a method or constructor of the AlgorithmHistory class.

**Note**: When reading in Nexus History record new execCount numbers are generated for the Parent AlgorithmHistory records. This should not need to change as the child AlgorithmHistory records should be allocated a default execCount number of 0.

## Recording the nested history records correctly

Currently Algorithms call a FillHistory method at the end of successfull execution, just before the workspace is stored. Prior to these changes ChildAlgorithms only store history if the AlwaysStoreHistory flag is set.

It is important to differentiate between WorkflowAlgorithms (which do not alter a workspace using anything other than other child algorithms) and other algorithms (that mix processing with possibly using other child algorithms). While it is desirable to include the child AlgorithmHistory records for WorkflowAlgorithms it is not adviable to do the same for all algorithms that use child algorithms. Take the LoadRaw algorithm for example: This uses two child Algorithms, LoadLog and LoadInstrument, however it also performs a lot of other changes to the workspace itself. If the included the child algorithm calls for this algorithm uses might expect they can run the detailed child algorithms to reproduce the effect of the parent algorithm, but it would fail badly and cause confusion.

Therefore we will only include Child Algorithms for WorkflowAlgorithms, or when the specific flag AlwaysStoreHistory is set.

### Changes to the Algorithm class

The current FillHistory() method performs two tasks:

It copies the history from all input workspaces into all output workspaces.

It add the history record for the current Algorithm to all output workspaces.

This needs to be refactored into three private methods. The first CopyInputWorkspaceHistory() should It copies the history from all input workspaces into all output workspaces, this should only ever be done for Parent workspaces (isChild=false), the values of AlwaysStoreHistory is irrelevant to this step.

The second method is new, DeleteChildTempWorkspaces(), the need for this is explained in the Workspace properties changes section below. This will scan over the workspace properties of and child algorithm records for \_\_TMP workspaces and add a DeleteWorkspace AlgorithmHistory record (to the end of the list) for each unique name found. This search need not be recursive as it will occur at each level.

The third method recordMyHistory () will perform differently depending on whether the algorithm is a parent or child (isChild).

|  |  |
| --- | --- |
| Parent (isChild=false) | Child (isChild=true) |
| Creates and stores the AlgorithmHistory record on all output workspaces. Including storing any child algorithm histories within the parent AlgorithmHistory class. | Nothing, unless AlwaysStoreHistory = true. Then contacts the Parent algorithm to store the child algorithm history. |

Workflow Algorithms will be identified by inheriting from a specific WorkflowAlgorithm abstract class that itself will inherit from Algorithm. This class will set the AlwaysStoreHistory flag to true by default for algorithms inheriting from it.

In order to allow child algorithms to contact their parents to store the history of the child it is necessary for child algorithm to hold a (weak) pointer to the parent algorithm. This will be set by the Algorith.createChildAlgorithm() method. It will be maintained as a weak pointer, and it’s validity must be tested before it is used (as it is possible that the parent algorithm has been aborted).

Finally the conditional execution of FillHistory() needs to be changed, currently it is only executed if IsChild=false or AlwaysStoreHistore=true. This needs to be changed to execute in all cases, as the conditional checks have moved within the refactored methods.

#### Python Algorithms

Currently Python Algorithms within other algorithms are not run as “proper” child algorithms, but are create as unmanaged algorithms. This will be changed to use the Algorithm.createChildAlgorithm() method. With this approach it should no longer be necessary to use the createChildAlgorithm() method directly within Python itself, instead any algorithms used within a python algorithm will automatically be created as child algorithms. However the method will be retained for historic reasons and to allow more detailed progress linkage.

Also the Workflow algorithm class will be exposed to python so it can be used just as the Algorithm class.

### Workspace properties changes

Currently there are two ways of setting a workspace property, one uses the name of a workspace in the ADS, and dynamically finds and provides a shared pointer to the workspace. This approach is fine, works well with history records and will not be changed.

The second approach is where you set the pointer to the workspace to be used, and do not provide a name (normally because it is not stored in the ADS). This approach is frequently used in child algorithms. At present this leaves the History record without any value for this property, and scripts will not run. We should change this to provide a temporary name for the workspace. I suggest a naming strategy as follows:

“\_\_TMP” + the memory address of the workspace (sptr.get())

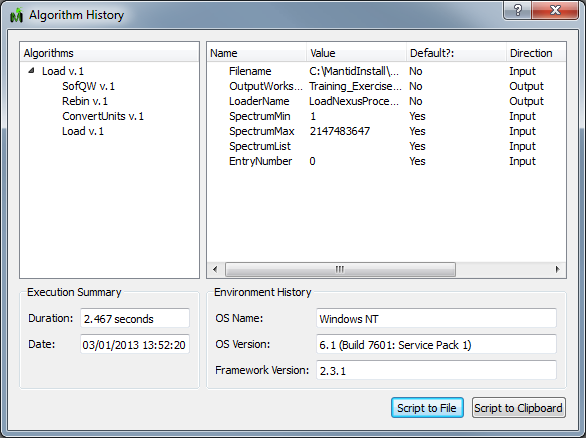
The final step would be to ensure these “\_\_TMP” workspaces are deleted from the ADS. This is done in the Algorithm. DeleteChildTempWorkspaces() method described earlier.

This should allow the underlying scripts of workflow algorithms to be run in their stead. However this must be tested!

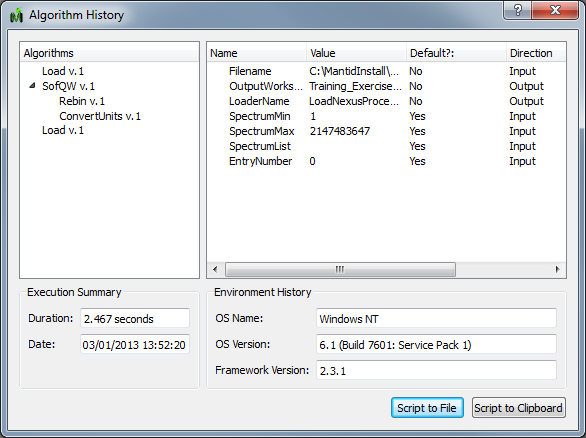
## Displaying and scripting the Nested history records

### WorkspaceHistory Display

This is the current Workspace History display.



I propose we make a few minimal changes to make it function for nested algorithm histories. The image is cropped to highlight the changes.



Specifically, change the algorithm display in the tree to:

1. Remove the artificial nesting under the most recent algorithm
2. Allow expansion under algorithms with child records.
3. Reverse the current order of entries in the list such that the most recent algorithm is at the bottom.
4. Upon opening the dialog should have only the Parent algorithms expanded, and the most recent selected.

### Python Scripting

Currently only parent algorithms are recorded in the Algorithm History (ignoring for now the special cases that use AlwaysStoreHistory). As such they never appear within the python scripts generated by the GeneratePythonScript algorithm.

This is a current generated python script:

######################################################################

#Python Script Generated by GeneratePythonScript Algorithm

######################################################################

Load(Filename='Data/CNCS\_7860\_event.nxs',OutputWorkspace='CNCS\_7860\_event')

ConvertUnits(InputWorkspace='CNCS\_7860\_event',OutputWorkspace='CNCS\_7860\_event',Target='DeltaE',EMode='Direct',EFixed='3')

Rebin(InputWorkspace='CNCS\_7860\_event',OutputWorkspace='CNCS\_7860\_event',Params='-2.9,0.03,2.9')

SofQW(InputWorkspace='CNCS\_7860\_event',OutputWorkspace='w1',QAxisBinning='0.2,0.2,2',EMode='Direct',EFixed='3')

Load(Filename='C:\MantidInstall\data\Training\_Exercise3a\_SNS.nxs',OutputWorkspace='Training\_Exercise3a\_SNS')

To include nested child algorithms I suggest we change this algorithm to output the following :

######################################################################

#Python Script Generated by GeneratePythonScript Algorithm

######################################################################

Load(Filename='Data/CNCS\_7860\_event.nxs',OutputWorkspace='CNCS\_7860\_event')

ConvertUnits(InputWorkspace='CNCS\_7860\_event',OutputWorkspace='CNCS\_7860\_event',Target='DeltaE',EMode='Direct',EFixed='3')

Rebin(InputWorkspace='CNCS\_7860\_event',OutputWorkspace='CNCS\_7860\_event',Params='-2.9,0.03,2.9')

SofQW(InputWorkspace='CNCS\_7860\_event',OutputWorkspace='w1',QAxisBinning='0.2,0.2,2',EMode='Direct',EFixed='3')

### Child Algorithms of SofQW – to run, remove the ‘#’ and comment out the SofQW line above ###

#ChildAlg1(InputWorkspace='CNCS\_event',OutputWorkspace='CNCS\_event')

#Child&ParentAlg(InputWorkspace='CNCS\_event')

#### Child Algorithms of Child&ParentAlg – to run, remove the ‘#’ and comment out the Child&ParentAlg line above ###

##ChildAlg2(Workspace=”\_\_TMP0x12345AE3456EF

##DeleteWorkspace(Workspace=”\_\_TMP0x12345AE3456EF

#### End of Child Algorithms of Child&ParentAlg ###

### End of Child Algorithms of SofQW ###

Load(Filename='C:\MantidInstall\data\Training\_Exercise3a\_SNS.nxs',OutputWorkspace='Training\_Exercise3a\_SNS')

This should not however alter the output for examples without workflow algorithms containing child algorithms.

Many facilities are now making use of event based data capture; This is already in use at the SNS at ORNL and is planned to be implemented at ISIS along with the next phase of instruments for TS2. Event based data allows the scientists much greater control over what data to include in their experiments and how to split it into periods. Crucially it also allows these decisions to be made, and changed after the experiment has completed.

The structure of this workspace within the workspace hierarchy is as follows, its inheritance from a Matrix Workspace is vital to allow its use in the majority of algorithms in Mantid.