Apollo

Data storage specification

confidential

Revision number: 1

Date:

Name: P. van der Velde

Contents

[Introduction 2](#_Toc208928698)

[In-memory data storage 2](#_Toc208928699)

[File data storage 3](#_Toc208928700)

# Disclaimer

This specification is not, by any stretch of the imagination, complete. It will need to be revised several times before it is complete. Currently several major parts are either missing or incomplete. This disclaimer will be updated to reflect any change in these sections. Finally a specification is supposed to be a ‘living’ document and therefore never complete. What were you thinking, better learn to live with this fact.

# Introduction

There are two main data storage parts, the in-memory storage and the file storage. Both have different goals. The in-memory storage should keep the data that is currently being processed safe and consistent. The file storage should keep the project data safe while it is not being used. Data written to file should be complete and enough to restore the original project. Furthermore the file should be resistant against damage while still taking up the minimal amount of space.

# In-memory data storage

**Goal:**

To provide access by the user and the application to the data that is currently in use. The data storage should allow users to manipulate that data while keeping the data safe and consistent. The data storage should also allow users to get back to an earlier state if this is desired, in both small steps (undo) and large steps (snapshot). It is however not a goal to protect the data from unexpected events like failure of the application or even power failures. In the case of such an event data loss is acceptable.

DATA STORAGE

* Make sure data is always consistent. Can use transactions (commands) or immutable data for this. Immutable data has the benefit that we can also provide undo’s
* It should be possible to write time based data as a series (good for moving meshes, unsteady sims etc.)
* It should be possible to write data as a series for undo-redo. This may mean that time-based data gets more complex? Should we store version numbers for each ‘change’?
* Should the data storage also store component info etc. or shall we separate that out into a different part?
* Allow data to be real data (data that can be used by components other than the generator) or ghost data (data that hasn’t really been committed, only usable by the generator and a visualizer). This is useful for suggested changes in e.g. the mesh generators etc.
* Data must have a state. Which can be:
  + Up to date
  + Invalidated (give info about why?)
  + Ghost (?)

UNDO / REDO

* Provide a limited (but large) number of undo’s? Or unlimited. Either way we do not store undo’s between sessions.
* Define what undo’s are for! While running we probably don’t need them and can just use snapshots but all the user actions should be undoable.
* Note that the undo’s also depend on the 3rd party system inside of which we’re running. We have to keep up with their undo’s! But only provide undo data if something has changed in our system. They could have many changes without us having any.

SNAPSHOTS

* Allow snapshots of the complete system. These should be able to be streamed to disk or other storage and contain a complete simulation / project. Users should be able to use these snapshots directly
* Allow the system to make snapshots automatically and return to a snapshot if things fail. Maintain a graph of snapshots (for undo purposes).
* The snapshot engine should be able to reset the simulation / project back to a specific snapshot but should ask components if any non-persistent data should be passed on to itself after restoring the snapshot (sort of information from the future). It should be possible to store this information with the next snapshot so that the ‘history + information from the future’ is safe. This is useful when we’re going back to a snapshot after a simulation crash. The information about the crash can be used to tweak the simulation, thereby hopefully preventing the crash. However we’ll need to store the information from the future so that we know what we have tried, in case things go wrong again.
* All data components that are persistent should support making snapshots + undo + transactions. These can all be defined through commands etc.
* Snapshots should be version resistant. Newer systems should be able to load older snapshots. Older systems are not required to load snapshots made by newer systems. However newer systems should ask the user if they want to save in the new or the old verion.
* A snapshot should be read-only. If the user wants to make changes then they’ll have to save the project / simulation separately.
* Snapshots should be robust. Corruption in the snapshot should be repairable
* Allow a snapshot to be copied to the head of the snapshot graph (for data resets)
* Should snapshots be stored in their own file or should they be stored inside the original file?
  + If we are storing snapshots in their own file how do we deal with users not saving the project and then running the simulation. What name will we give the snapshots and will we rename the snapshots when we save the file.
  + If we are storing the snapshots in the project file how do we store the snapshots if the user has not saved the file? Do we force the user to save the file before they can run the simulation?

# File data storage

* Store CRC’s etc. to make the file robust against corruption.