HPS-MC status and plans

Insights and opinions of an HPS novice

Sarah Gaiser

October 18, 2022

U.S. DEPARTMENT OF ENERGY | Stanford University

SLAC NATIONAL ACCELERATOR LABORATORY

1. First impressions
2. Implemented changes
3. Ideas for future updates
4. Current project
5. Summary

First impressions

First impressions

- Overall: HPS software very confusing
  - Distributed over several programs/repositories
  - No or not easily accessible documentation
  - In parts: code badly formatted, not readable
- No (complete) guidelines on how to get started
  - Exception: installation guides exist
  - New people very dependent on older collaboration members $\rightarrow$ stuck if those not available

Implemented changes – Documentation

SLAC

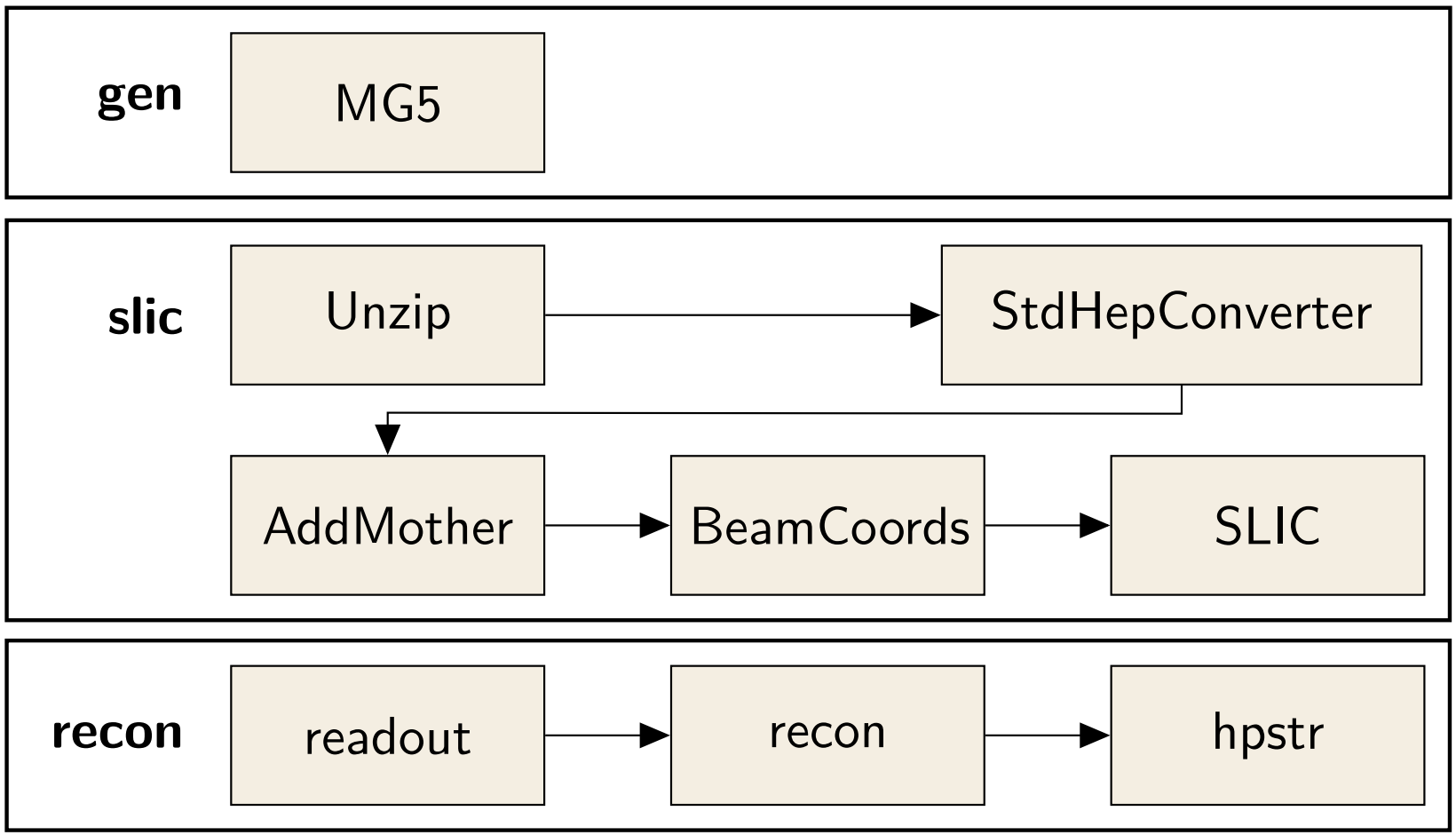
- Doxygen = software tool to generate browsable documentation from comments in code
- Documentation for hps-mc, slic, and hpstr
  - Doxygen combined with GitHub actions
  - Automatically updated and deployed on the corresponding website on every merge
- Doxygen requires a specific comment style
  - Documented on Doxygen website
  - Refer to existing comments in hps-mc, slic, and hpstr for further guidance
- See also: How to install Doxygen and Setup Doxygen for code documentation

Implemented changes – Examples and prod

SLAC

- hps-mc/examples: easily executable and fast scripts that illustrate the different use cases for hps-mc
  - For some examples: automatically download files to run tests
    - Files are stored at JLab (same area as hps-java test data)
    - Job script will recognize url and download file using wget to scratch directory
- hps-mc/prod: new space in hps-mc to streamline the production of MC data
  - Separate data generation in three stages: generation (gen), slic, readout and reconstruction (recon)
  - So far: A-prime, beam, fee, tritrig, and tritrig+beam

Example production workflow



Updates to hps-mc

SLAC

Implemented changes

- Code documentation
  - Explanation of classes, variables, functions, etc.
  - Short written introductions of functionality and structure
- hps-mc/examples: short example scripts
  - Not all examples were functional
  - Additional instructions and explanations

Updates to hps-mc

SLAC

Implemented changes

- Code documentation
  - Explanation of classes, variables, functions, etc.
  - Short written introductions of functionality and structure
- hps-mc/examples: short example scripts
  - Not all examples were functional
  - Additional instructions and explanations

To do

- Overview of HPS software: hps-mc, hpstr, slic, hps-java, ...
  - Which program is responsible for what?
  - How are different things connected?
  - When/how do I use the different components?
- Uniform code formatting
- Unit and functional testing

Code formatting

SLAC

- General idea: implement linter in CI pipeline to ensure good code formatting
- What tool do we want to use?
  - What are our requirements?
    - Compatibility with Doxygen style comments
    - ...
  - Which python style do we want to follow?
    - Personally, no strong opinion $\rightarrow$ want to start discussion
  - Possible tool: pycodestyle – configurable to work with Doxygen style comments
- How would it work?
  - When pushing/merging code, format is checked and mistakes highlighted
  - Formatting by hand or using tools like autopep8
  - If issues are fixed, process as usual

Functional and unit testing

SLAC

- Functional tests
  - Run program and compare result to expected outcome
  - Black-box tests: no information on internal workings of program
- Unit tests
  - Used to validate single units of code
  - White-box tests: ensure correctness of code functions and code-snippets

Functional and unit testing

SLAC

- Functional tests
  - Run program and compare result to expected outcome
  - Black-box tests: no information on internal workings of program
- Unit tests
  - Used to validate single units of code
  - White-box tests: ensure correctness of code functions and code-snippets
- Why is this important?
  - Catch bugs and logical mistakes easily and early on
  - Reduces overall development time if test written in parallel to production code
  - Simplfy code review and general understanding of code functionality
  - Testable code usually follows good coding practices/principles

Functional and unit testing

SLAC

How can we make this work?

- Problems
  - Code not completely testable at the moment
  - Takes time to write tests
  - Many programming languages $\rightarrow$ many testing frameworks
- Possible solutions
  - Testability: write tests for testable code, refacture or ignore the rest
  - Time: Use test-writing as tool to understand code more deeply
    - *Viele Hände, schnelles Ende*
    - github copilot: automatic generation of unit tests
  - Testing frameworks:
    - C++: googletest
    - Python: unittest (?)
    - Java: JUnit (?)

Current project – Alignment studies

SLAC

- Goal: understand effect of misalignments on data
- At the moment: simulate, reconstruct, and analyze with two different detector versions
  - Complete pipeline with one detector
  - Simulate with one, reconstruct with the other
  - Compare different versions
- Other ideas:
  - Analyze effect of changed distance between sensors and randomly distributed Tu and Rw for each sensor
  - Determine reasonable widths of normal distributions
  - Create sw tools for quickly generating new detectors
- In the meantime: document steps and create short how-to's for (personal) reference

Summary

SLAC

- Successfully implemented changes
  - Code documentation with Doxygen
  - Cleanup of hps-mc/examples
  - Setup of MC production pipeline in hps-mc/prod
- Things we need
  - Good introduction to and overview of HPS software
  - Better (=uniform) code formatting
  - Eventually: checks for code quality and correctness; structural and unit tests