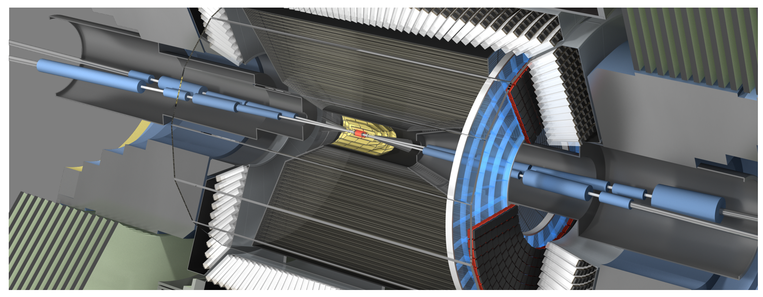
Belle II Documentation Effort using Sphinx

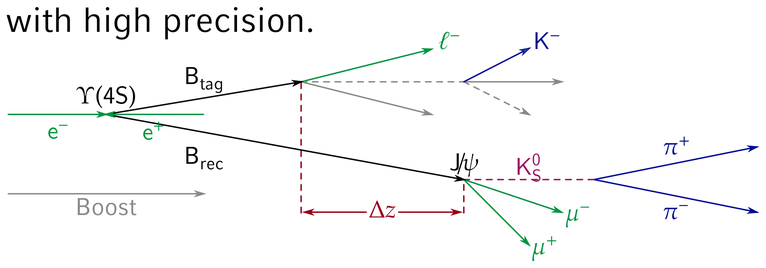


23rd International Conference on Computing in High-Energy and Nuclear Physics
Software Development

M. Ritter[1], C. Pulvermacher[2], I. Komarov[3], T. Hauth[4] and T. Kuhr[1] for the Belle II Collaboration
[1]LMU, Munich, Germany [2]KEK, Tsukuba, Japan, [3]DESY, Hamburg, Germany, [4]KIT, Karlsruhe, Germany

The Belle II Experiment

An electron positron collider with asymmetric energies located in Japan to test the standard model with high precision.

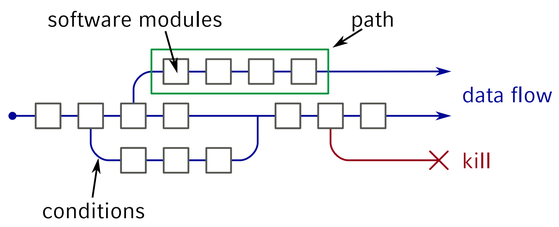


- start in 2018, collect 50 ab^{-1} until 2024
- record 4×10^{11} events, 60 PB of data

Software Framework

Software framework written from scratch using experience from Belle and other HEP experiments.

- core framework implemented in C++14 and including the boost libraries
- use ROOT 6 framework for serialization of event data, Geant4 for simulation
- Python 3 interface for configuration and high level program steering
- different algorithms (called modules) are executed sequentially for each event



Analysis Concept

For analysts we provide high level constructs to work in a candidate based analysis scheme

- text based cuts on “variables”
- basically no C++ for users

```
# create Ks -> pi+ pi- list from V0
# keep only candidates with 0.4 < M(pipi) < 0.6 GeV
fillParticleList('K_S0:pipi', '0.4 < M < 0.6')

# reconstruct J/psi -> mu+ mu- decay
# keep only candidates with 3.0 < M(mumu) < 3.2 GeV
reconstructDecay('J/psi:mumu -> mu+:loose mu-:loose',
                 '3.0 < M < 3.2')

# reconstruct B0 -> J/psi Ks decay
# keep only candidates with 5.2 < M(J/PsiKs) < 5.4 GeV
reconstructDecay('B0:jspiks -> J/psi:mumu K_S0:pipi',
                 '5.2 < M < 5.4')

# perform B0 kinematic vertex fit using only the mu+ mu-
# keep all candidates (C.L. of fit >=0)
vertexRave('B0:jspiks', 0.0, 'B0 -> [J/psi -> ^mu+ ^mu-] K_S0')

# build the rest of the event associated to the B0
buildRestOfEvent('B0:jspiks')

# perform MC matching (MC truth asociation). Always before TagV
matchMCTruth('B0:jspiks')

# calculate the Tag Vertex and Delta t (in ps)
# breco: type of MC association.
TagV('B0:jspiks', 'breco')

# save candidates to ntuple
variablesToNtuple("B0:jspiks", ["Mbc", "dE", "DeltaT"])
```

➡ Need to document custom concepts for Users

- software modules and their parameters
- python functions
- variables (and parameters)

Documentation Tools

Usually documentation is build automatically (doxygen) or completely manually (wiki style)

Automatic systems

- Often impose rigid structure
- Written for one language, hard to extend

Manual documentation

- Large overhead
- Prone to be outdated

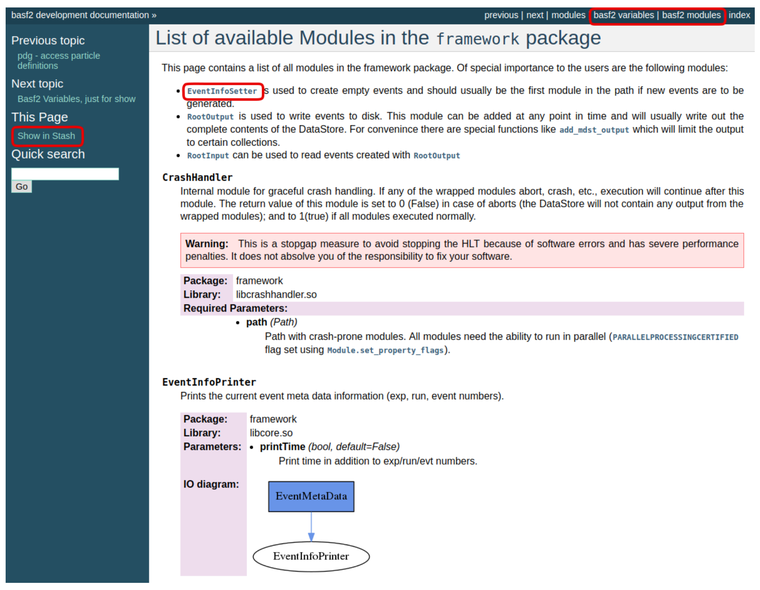
SPHINX
Python Documentation Generator

Originally created for the Python documentation

- `http://sphinx-doc.org`
- Written in Python
- Multiple output formats (HTML, LaTeX, …)
- Hierarchical structure
- Extensive cross referencing
- Cross language support
- Easily extendible

➡ Used to create one consistent user manual

- Sweet spot between manual and automatic
- Fits well into our Python based workflow


basf2 development documentation » | previous | next | modules | basf2 variables | basf2 modules | index

Previous topic: pdg - access particle definitions
Next topic: Basf2 Variables, just for show
This Page: Show in Stash
Quick search: Go

List of available Modules in the `framework` package

This page contains a list of all modules in the framework package. Of special importance to the users are the following modules:

- `EventInfoSetter` is used to create empty events and should usually be the first module in the path if new events are to be generated.
- `RootOutput` is used to write events to disk. This module can be added at any point in time and will usually write out the complete contents of the DataStore. For convenince there are special functions like `add_mdst_output` which will limit the output to certain collections.
- `RootInput` can be used to read events created with `RootOutput`

`CrashHandler`
Internal module for graceful crash handling. If any of the wrapped modules abort, crash, etc., execution will continue after this module. The return value of this module is set to 0 (False) in case of aborts (the DataStore will not contain any output from the wrapped modules); and to 1(true) if all modules executed normally.

Warning: This is a stopgap measure to avoid stopping the HLT because of software errors and has severe performance penalties. It does not absolve you of the responsibility to fix your software.

Package: framework
Library: libcrashhandler.so
Required Parameters:
- **path** *(Path)*
  Path with crash-prone modules. All modules need the ability to run in parallel (`PARALLELPROCESSINGCERTIFIED` flag set using `Module.set_property_flags`).

`EventInfoPrinter`
Prints the current event meta data information (exp, run, event numbers).

Package: framework
Library: libcore.so
Parameters:
- **printTime** *(bool, default=False)*
  Print time in addition to exp/run/evt numbers.

IO diagram: EventMetaData → EventInfoPrinter

Custom Extensions

We developed custom Sphinx extensions to allow documentation of all our user concepts

- Easy extraction of documentation and parameters of software modules and variables interfaced in Python
- Automated data flow charts from code
- Document logical groups instead of purely alphabetical
- Automatic index creation for alphabetic list
- Automatic links to Python documentation

➡ fully integrated and searchable documents of all our tools

Documentation close to the Code

Developers can document directly in the code

- Completely integrated in build system
- Easy syntax, human readable also in code or textbased output
- Can be enforced in pull requests

```
REGISTER_VARIABLE("myvariable(arg1, arg2)", functionPtr, R"DOC(
Description of the variable
in as many lines as necessary.

See also:
   :b2:var:`someOtherVariable`

Parameters:
   arg1: The first argument of the function
   arg2: And some other argument
)DOC");
```

➡ Provide user friendly documentation evolving with the code

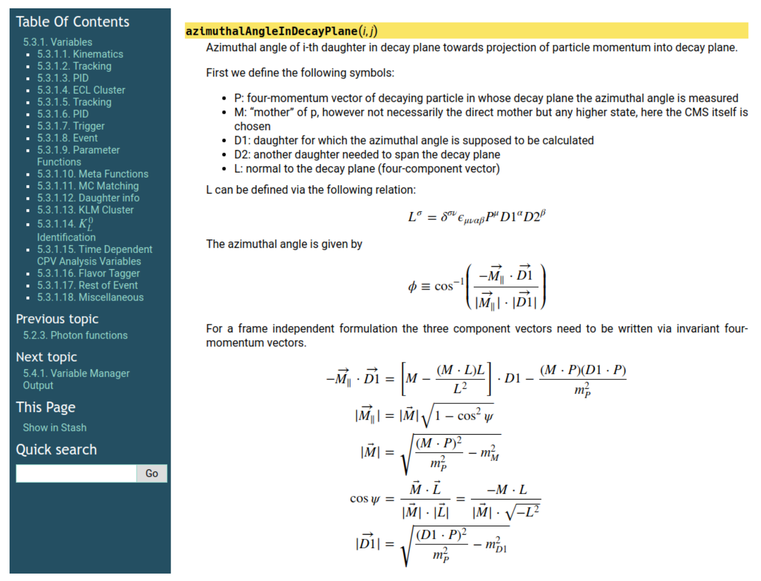


Table Of Contents
- 5.3.1. Variables
  - 5.3.1.1. Kinematics
  - 5.3.1.2. Tracking
  - 5.3.1.3. PID
  - 5.3.1.4. ECL Cluster
  - 5.3.1.5. Tracking
  - 5.3.1.6. PID
  - 5.3.1.7. Trigger
  - 5.3.1.8. Event
  - 5.3.1.9. Parameter Functions
  - 5.3.1.10. Meta Functions
  - 5.3.1.11. MC Matching
  - 5.3.1.12. Daughter info
  - 5.3.1.13. KLM Cluster
  - 5.3.1.14. K_L^0 Identification
  - 5.3.1.15. Time Dependent CPV Analysis Variables
  - 5.3.1.16. Flavor Tagger
  - 5.3.1.17. Rest of Event
  - 5.3.1.18. Miscellaneous

Previous topic: 5.2.3. Photon functions
Next topic: 5.4.1. Variable Manager Output
This Page: Show in Stash
Quick search: Go

`azimuthalAngleInDecayPlane`(*i,j*)
Azimuthal angle of i-th daughter in decay plane towards projection of particle momentum into decay plane.

First we define the following symbols:

- P: four-momentum vector of decaying particle in whose decay plane the azimuthal angle is measured
- M: “mother” of p, however not necessarily the direct mother but any higher state, here the CMS itself is chosen
- D1: daughter for which the azimuthal angle is supposed to be calculated
- D2: another daughter needed to span the decay plane
- L: normal to the decay plane (four-component vector)

L can be defined via the following relation:

$$L^\sigma = \delta^{\sigma\nu}\epsilon_{\mu\nu\alpha\beta}P^\mu D1^\alpha D2^\beta$$

The azimuthal angle is given by

$$\phi \equiv \cos^{-1}\left(\frac{-\vec{M}_\parallel \cdot \vec{D1}}{|\vec{M}_\parallel| \cdot |\vec{D1}|}\right)$$

For a frame independent formulation the three component vectors need to be written via invariant four-momentum vectors.

$$-\vec{M}_\parallel \cdot \vec{D1} = \left[M - \frac{(M\cdot L)L}{L^2}\right]\cdot D1 - \frac{(M\cdot P)(D1\cdot P)}{m_P^2}$$
$$|\vec{M}_\parallel| = |\vec{M}|\sqrt{1-\cos^2\psi}$$
$$|\vec{M}| = \sqrt{\frac{(M\cdot P)^2}{m_P^2} - m_M^2}$$
$$\cos\psi = \frac{\vec{M}\cdot\vec{L}}{|\vec{M}|\cdot|\vec{L}|} = \frac{-M\cdot L}{|\vec{M}|\cdot\sqrt{-L^2}}$$
$$|\vec{D1}| = \sqrt{\frac{(D1\cdot P)^2}{m_P^2} - m_{D1}^2}$$

Migration from Doxygen

Existing code might have doxygen style documentation

- Sphinx has callback mechanism before emitting documentation
- Allows different documentation styles (Google/Numpy style docstrings)
- Custom callback to convert existing Doxygen docstrings as well

Documentation of C++ Code

Currently our Sphinx documentation contains almost no C++. In the future we also plan to move the developer documentation to Sphinx.

- Sphinx has full support for C++ documentation including cross referencing
- Sphinx can utilize doxygen xml output via “Breathe” extension
- Classes could be directly documented using cling/libclang
- Combine Sphinx & doxygen: keep short, user friendly interface documentation in Sphinx and link to full listing in doxygen

➡ Full control over documentation details

Belle II
Excellence Cluster Universe
Bundesministerium für Bildung und Forschung