Analysis software.

Introduction, design, some new features, and tips

Sam Cunliffe
Bonn-DESY-Göttingen Inclusive Meeting, Hamburg, 13.11.2019
My plan for this talk

● The Belle II software.
  ▪ packages, Modules, Paths, DataStore, database.

● A brief (hopefully useful) diversion: light releases

● The analysis package.
  ▪ Design.
  ▪ Particles, candidates.
  ▪ New features of interest to inclusive/missing energy analyses.

● Misc. tips and advice.
Where to go for help/examples

Slightly less flippant

• **Software doc:** software.belle2.org
  ▪ **Here** is what’s new in release-04 (or just go there and search).

• **Q&A:** questions.belle2.org.

• **search.belle2.org.**

• **Code:** stash.desy.de (the B2 “project” is basf2).

• **Tutorials:**
  ▪ **Here** is the indico category for past starterkit workshops.
  ▪ Interactive tutorials: jupyterhub.belle2.org.
  ▪ $BELLE2_RELEASE_DIR/analysis/examples/*
basf2: packages, Modules, Paths, DataStore, and the database
What?

**Comput Softw Big Sci (2019) 3: 1.**

- basf2 = Belle II analysis software framework.
  - Why not b2asf? I don’t know. Nobody consulted me.

- Don’t be fooled by the word “analysis”.
  - Actually basf2 does everything. You should call it “The Belle II core software”.
  - Analysis is a subset of basf2.

- It’s also a **python module** for steering core functionality.

- It’s also the name of the **executable**.
  - … which can be thought of as ipython3.
Packages

• Code is organised into directories (== packages).
  ▶ Code for similar tasks lives together.
  ▶ There is one for each subdetector, and one for tracking, simulation, etc.

• As a postdoc/PhD student you might do some development in another package.
  ▶ But as an analyst, you really only care about a few:

  - Code to do an analysis and "steer" the analysis.
  - The "user interface".
  - Multivariate analysis (classifier training, machine learning)
  - Configuration files to generate your decay.
  - High level scripts to "skim" data to get it into a usable size: run centrally.
  - Controlled basf2 data format.
Package structure

Some directories are special

- Build system is called **scons**.
  - It’s quite cool.
- Some special/important directories within a package.
  - **scripts** (for python code… not scripts)
  - **examples** (for example scripts)
  - **modules** (compiled by the build system)
  - **dataobjects** (compiled by the build system)
  - **dbobjects** (database objects; compiled by the build system)
  - **tests** (executed automatically in pull requests and nightly)
  - **tools** (executables, added to $PATH should be named b2something-doer)
  - **variables**
- All of this will become clear or (I hope) is self-explanatory.
What do we need to process the data?

1) A set of classes (modules) that process the data
   \[\rightarrow\] basf2 Module
Modules, Paths, the DataStore

What do we need to process the data?

1) A set of classes (modules) that process the data
   → basf2 Module
Modules, Paths, the DataStore

What do we need to process the data?

1) A set of classes (modules) that process the data
   → basf2 Module

2) Classes that hold data and allow module to pass thing one to the other
   → basf2 dataobject (lives in the datastore)
What do we need to process the data?

1) A set of classes (modules) that process the data
   → basf2 Module

2) Classes that hold data and allow module to pass thing one to the other
   → basf2 dataobject (lives in the datastore)

3) An order in which the modules must be executed
   → basf2 Path
The database

For data-taking / simulation “conditions”

4) The database
What is a Global Tag?

payload #1  
  rev.1   rev.2   rev.1
payload #2  
  rev.1   rev.2
payload #3  
  rev.1   rev.3

payload  One atom of conditions data (e.g. BeamParameters). In our case this is just a file. They are identified by name and have different revisions

IOV  Short for "interval of validity", the run interval for which the payload is valid. Can be a fixed run range (closed) or starting at a given run (open)

global tag  Is an immutable set of payloads and their IOVs
A word on dataobjects and Belle II file types

This sounds boring but it’s quite important

- All basf2 dataobjects are Belle2::RelationsObjects (inherits from ROOT::TObject).
  - But you should not try to open files with root.
  - “Kind-of” works. But totally unsupported, absolutely no guarantee of backward (or any) compatibility.

- dst: can save any of the dataobjects in the DataStore (with EStoreFlags::c_WriteOut)
  - data summary table.

- mdst: the legacy/official analysis format.
  - mini-dst.
  - Strictly controlled list of dataobjects.
  - Has МЯ implications in computing budget.

- cdst: calibration format.
  - mdst + a few others.
  - Huge, so only produced sparingly and for calibration.

- udst: user format. mdst + Belle2::Particles.

```python
branches = [ 'Tracks', 'V0s', 'TrackFitResults', 'EventLevelTrackingInfo', 'PIDLikelihoods', 'TracksToPIDLikelihoods', 'ECLClusters', 'ECLClustersToTracksNamedBremsstrahlung', 'EventLevelClusteringInfo', 'TracksToECLClusters', 'KLMClusters', 'KLIds', 'KLMClustersToKLIds', 'TRGSummary', 'SoftwareTriggerResult', ]
```

$BELLE2_RELEASE_DIR/mdst/scripts/mdst.py
Jargon

A quick resume

- **basf2**: The Belle II software / a **python** module / an executable.
- **Package**: related code lives together.
- **Module**: does a data processing task.
- **Path**: ordered list of modules.
- **DataStore**: place where the **dataobjects** live. Written to and read from by modules.
- **Database**: place where conditions data lives, in the form of **payloads**.
  - Modules can access it.
- **Global tag**: a labeled collection of database data.
  - Historically typed by the user into her scripts.
  - Now not needed*.
- **mdst**: the Belle II data file format for analysis / a package containing the dataobjects.

*) unless you want to run the FEI
Distributed resources (GRID)

Centrally managed

- mdst
- udst

Reconstruction

RAW

MC production

Analysis users

- Official skim
- basf2 analysis
- User skim

Ntuple

Local resources

Download from grid

Ntuple

Offline analysis (pandas, local ROOT)
Light releases

Are cool

- Subset of packages that compile standalone:
  \{ \texttt{analysis}, \texttt{mdst}, \texttt{mva}, \texttt{skim},
  \texttt{framework}, \texttt{geometry} \} 

- Everything that is needed to open mdst
  $\Leftrightarrow$ everything needed for analysis.

- Make light releases often $\rightarrow$ faster development cycle than full \texttt{basf2}.

- More flexible. Can be powerful.
  - e.g. FEI with baryonic modes is not available in release-04 but it is in the light release.
Limitations

You can’t generate (you should not be doing that anyway)

- Generation / simulation / reconstruction jobs won’t work (by construction).
  → `ModuleNotFoundError` s if you try.
- `cdst` not supported (by construction)
- Semver major release.
  - Backward compatibility is not guaranteed.

Only supported actions:
- Read a preexisting mdst file.
- Make particles and do analysis things (vertex fits, FEI, TreeFitter).
- Write out user ‘tuples
Analysis package:
Particles, variables
Particles

- **Candidate** `Belle2::Particle` created from mdst dataobjects or composite:
  - Track w/ PIDLikelihood → Charged particle.
  - Calorimeter cluster → Neutral particle.

- Core object: `Belle2::ParticleList` of all such candidates in an event.

- Combine `Belle2::ParticleLists` with the `ParticleCombiner` (a module) to create *composite* particle candidates (e.g. $B \rightarrow K_S \pi^0\gamma$).
Analysis tools

- High-level tools operate on `Belle2::ParticleLists` (once per candidate).
  - Fit vertex (RAVE, Kfit).
  - Kinematic fitting (Kfit, fork of ILC/Marlin).
  - Tag the B flavour.
  - Build the RestOfEvent.
  - Write candidates for offline study.
  - Fit full decay chain.
- Notable exceptions:
  - EventShape.
  - Full event interpretation.

Artist's impression of event shape: U. Tamponi
A typical path for an analysis job

```
DataStore
```

```
mdst.root
```

```
histos.root
```

```
Load mdst
mdst objects → final state particle lists
Combine particles
Do some selection
Do some vertex fitting
Calculate complicated variables
Do some more selection
Write info about selected particles
```

```
mdst objects
ParticleList
new ParticleList(s)
vertices
variables dataobject
final list of candidates
```

```
selected ParticleList
```

```
selected ParticleList
```

```
A typical path for an analysis job
```

```
Analysis software | S Cunliffe, 13.11.2019
```
A typical path for an analysis job

Now with the real names for the modules

DataStore

RootInput

ParticleLoader

ParticleCombiner

ParticleListManipulator

ParticleVertexFitter

ContinuumSuppression

ParticleListManipulator

VariablesToTtuple

mdst.root

histos.root

mdst objects

ParticleList

new ParticleList(s)

selected ParticleList

vertices

variables dataobject

final list of candidates

Analysis software | S Cunliffe, 13.11.2019
How to load a module

- The syntax is the same no matter what module you want.
- You “register” it, (optionally) set some parameters, then add it to the path.

```python
mod = basf2.register_module('ModuleName')
mod.param('someParameter', value)
mypath.add_module(mod)
```
Can you figure out what this does?

```python
pcomb = basf2.register_module('ParticleCombiner')
pcomb.param('decayString', 'K*0:myKst -> K+:good pi-:good')
pcomb.param('cuts', '0.6 < M < 1.0')
path.add_module(pcomb)
```
Can you figure out what this does?

- If you said “combine particles” you’re close to correct.
- Actually it adds a module which **will do that** to the path.

```python
pcomb = basf2.register_module('ParticleCombiner')
pcomb.param('decayString', 'K^0:myKst -> K+:good pi^-:good')
pcomb.param('cuts', '0.6 < M < 1.0')
path.add_module(pcomb)
```

- Actually this is a common misconception.
- We are not in an event loop.
- We are configuring the processing we want to happen to our events.
Shorthand function syntax

reconstructDecay just configures a ParticleCombiner

- Module setup can get complex.
- There are convenience functions. But this is a double-edged sword.

```python
pcomb = basf2.register_module('ParticleCombiner')
pcomb.param('decayString', 'K*0:myKst -> K+:good pi-:good')
pcomb.param('cuts', '0.6 < M < 1.0')
path.add_module(pcomb)
```

```python
modularAnalysis.reconstructDecay(
    'K*0:myKst -> K+:good pi-:good', '0.6 < M < 1.0', mypath)
```
A typical path for an analysis job:

```python
import basf2
from modularAnalysis import inputMdst

mypath = basf2.Path()
inputMdst('default', '/path/to/input.mdst.root', path=mypath)
# usually no need to set the global tag in release-04
```
from modularAnalysis import fillParticleList

good_tracks_p3 = 'abs(dr) < 2 and abs(dz) < 0.5'
# https://confluence.desy.de/x/zJv-Bw

fillParticleList('K+:good', good_tracks_p3, path=mypath)
fillParticleList('pi+:good', good_tracks_p3, path=mypath)
from modularAnalysis import reconstructDecay
reconstructDecay(
    'K*0:myKst -> K+:good pi-:good', '0.6 < M < 1.0', path=mypath)
A typical path for an analysis job

```
from modularAnalysis import applyEventCuts
applyEventCuts('nTracks < 10', path=mypath)
# see also nCleanedTracks (probably better)
```
A typical path for an analysis job

from vertex import fitVertex
fiteVertex('K*0:myKst', conf_level=0.0, path=mypath)
A typical path for an analysis job

```python
from modularAnalysis import buildRestOfEvent
buildRestOfEvent('K*0:myKst', path=mypath)
```
A typical path for an analysis job

from modularAnalysis import cutAndCopyLists
cutAndCopyLists('K^0:hiP', 'K^0:myKst', 'p > 1.0', path=mypath)
A typical path for an analysis job

```python
from modularAnalysis import variablesToNtuple
variablesToNtuple('K*0:hiP', ['p', 'M'], path=mypath)
```
A typical path for an analysis job

# now we actually run the event loop!
basf2.process(mypath)
print(basf2.statistics)
The VariableManager

It manages variables

• The place where variables are calculated and evaluated. C++.
• Contains a lot of things that you might be interested in.
• Easily extensible (but what you want probably already exists).
• Many newcomers first point of entry into the code is tweaking or adding a variable.
  ▶ We are really picky about clear documentation.
• Used for selections. And you’ve already seen it.
  ▶ M, E, p, theta, phi, nTracks, …
  ▶ clusterE, nChargeZeroTracks, nParticlesInList(), totalEnergyOfParticlesInList(), cosAngleBetweenMomentumAndVertexVector, …
6.3. Variables

While basf2 operates on ParticleList, it is also important to calculate physics quantities associated with a given candidate or event.

In basf2 analysis, variables are handled by the VariableManager. There are many variables available for use in analysis. Probably the most obvious, and useful are: \( p \), \( E \), \( M_{bc} \), and \( \Delta E \).

You can search the variables in an alphabetical Basf2 Variable Index, or browse Variables by group.

- 6.3.1. VariableManager
- 6.3.2. Variables by group
  - Kinematics
  - Helicity
  - Tracking
  - PID
  - Basic particle information
    - PID for expert
  - ECL Cluster
  - Acceptance
  - Trigger
Aliases

Are awesome and you should use them

- I overhauled and cleaned up the documentation for the VariableManager.
- One thing worth mentioning: aliases.

```python
from variables import variables as vm
vm.addAlias('cosPVtx', 'cosAngleBetweenMomentumAndVertexVector')
```

There is no path here!
The VariableManager exists alongside the path
New features of interest to inclusive and missing energy analyses
Mc matching for inclusives

Yo Sato, SC; BII-4463

- Helper variables for sum-of-exclusive and fully inclusive mc-matching.
- Implemented new grammar: “@X”.
- Deleted some unused grammar defined in DecayString.
- Plan to write a BELLE2-NOTE.

```python
from modularAnalysis import reconstructDecay, copyLists
reconstructDecay('@Xsd:0 -> K_S0:all', cuts, path=mypath)
reconstructDecay('@Xsd:1 -> K+:loose pi-:loose', cuts, path=mypath)
reconstructDecay('@Xsd:2 -> K+:loose pi-:loose pi0:all', cuts, path=mypath)

copyLists('Xsd:all', ['Xsd:0', 'Xsd:1', 'Xsd:2'], path=mypath)
reconstructDecay('B0:Xdmmumu -> Xsd:all mu+:loose mu-:loose', cuts, path=mypath)
```
ROEs and superparticles

Sviat Bilokin, Frank Meier, Will Sutcliffe, Peter Lewis; BII-4995

- Actually treat the ROE like a full `Belle2::Particle`.

- Useful in some missing energy analyses.
  - A new `AllParticlesCombinerModule`.
  - An ROE particle implementation (needs testing).

- Also: provide an official, supported solution for, e.g. vertex-fitting, with neutrinos (or other missing particles).

```cpp
reconstructDecay("vpho:beam -> B+:sig ROE", "", mypath)
```
VariablesToHDF5

Martin Ritter, BII-3808

• “Just” a new module (python, for the moment).

• Output to HDF5 file containing a pytable.
  ▶ Easily read in a `pandas.DataFrame`.

• Won’t immediately work on the grid (file extension will not be .root).

```python
from b2pandas_utils import VariablesToHDF5
v2hdf5 = VariablesToHDF5(
    filename="variables.hdf5", listname="pi+:all",
    variables=list_of_interesting_event_variables)
path.add_module(v2hdf5)
```
Misc. tips
Misc. tips

And common pitfalls

- Caution with global event calorimeter information.
  - Clusters are not unique. The energy depends on the hypothesis.
  - The number of clusters / total energy is background dependent.
  - Make a simple selection and count/sum up “good” photons.

- Global event KLM information.
  - Don’t even.

- Use the ParticleLoader and use Belle2::Particles / Belle2::ParticleLists.
  - Raw mdst dataobjects are subtle, complicated and the probability of making a mistake → 1.
  - Analysis code shouldn’t really need to interact with them directly.

- Don’t open *dst files in root.

- You should not need to set a global tag in release-04. If you do something is probably wrong.

- Read the convenience functions in modularAnalysis.

- Complain (loudly, via bug reports) about missing documentation.
Misc. tips
And common pitfalls

- Caution with global event calorimeter information.

from variables import variables as vm
from modularAnalysis import fillParticleList

good_photons = 'clusterNHits > 1.5 and E > 0.05'
fillParticleList('gamma:good', good_photons, mypath)
vm.addAlias('smartEECLNeutral',
            'totalEnergyOfParticlesInList(gamma:good)')

- You should not need to set a global tag in release-04. If you do something is probably wrong.

- Read the convenience functions in modularAnalysis.

- Complain (loudly, via bug reports) about missing documentation.
software.belle2.org
questions.belle2.org
Stuff in these slides profited from ideas and revisions by these people:

Jake Bennett, Umberto Tamponi, Hannah Wakeling, and Anže Zupanc
The big picture

Published paper

What do you need?

Nature

PYTHIA
The big picture

What do you need?

Data storage → root (trees) files
Data processing → C++ code
Scripting → Python
The big picture

Nature

Real Detector

rawData.root

PYTHIA

Detector Simulation GEANT4
The big picture

Nature

Real Detector

PYTHIA

Detector Simulation GEANT4

Reconstruction

rawData.root

ECLClusters, Tracks, PIDLikelihoods

(c,m)dst.root

basf2 analysis steering script

nTuples.root

MCParticles
The big picture

- Nature
- PYTHIA
- Detector Simulation GEANT4
- Real Detector
- Reconstruction
- rawData.root
- (c,m)dst.root
- ECLClusters, Tracks, PIDLikelihoods
- basf2 analysis steering script
- nTuples.root
- MCParticles
- Published paper
- Analysis note
- Measurement
- “Offline” analysis

Analysis software | S Cunliffe, 13.11.2019
The big picture

- Nature
- PYTHIA
- Detector Simulation GEANT4
- Real Detector
- rawData.root
- Reconstruction
- (c,m)dst.root
- ECLClusters, Tracks, PIDLikelihoods
- Analysis note
- Measurement
- Happy supervisor
- cool things with pandas
- nTuples.root
- basf2 analysis
- steering script
- Published paper
- Analysis note
- Measurement
- rawData.root
- ECLClusters, Tracks, PIDLikelihoods
- (c,m)dst.root
- Analysis note
- Happy supervisor
- cool things with pandas
- nTuples.root