Wrapped Legacy Reconstruction

Bartosz Maksiak

2020-11-04

Table of Contents

Running reconstruction

Basic example

While on LxPlus

1. Set environment variables (32-bit version!)
2. Copy files from *apps/Standard/Reconstruction* to the local directory
3. Copy input RAW file to the local directory
4. Run the script *runModuleSeq.sh* with the proper parameters

Parameters of the *runModuleSeq.sh* script

- *-i* – path to input RAW file
- *-o* – output filename (*.shoe.root* suffix will be added automatically)
- *-b* – path to *bootstrap.xml* file
- *-v* – vertex fitting (*pp* for long targets, *pA* for thin targets)
- *-k* – global key (global keys browser available here: https://www.nuph.us.edu.pl/hepdb/)

Calibration database and local keys

Calibration database in an essential element of proper data reconstruction. It holds calibration history of the detector elements.

Every element of the detector has its one or several sub-elements that need to be calibrated during calibration process. Examples:

- TPC: geometry, drift velocity, chamber T0, global T0, ...
- BPD: geometry, strip gains
- Target: geometry
- ...

Each of these sub-elements has its own calibration that may change with time (after multiple iterations of calibration). Every version of calibration is stored there – it is called *local key* (e.g. local key V17A of the TPC geometry, local key V16D of BPD strip gains).

Calibration database and global keys

Global key is a set of local keys for all detector elements. For example: **15_043** is set of BPD geometry local key **V15K**, TPC alignment geometry local key **V13I**, Target geometry local key **V15M**, and so on.

So, by setting particular global key we choose automatically settings for particular detector elements calibrations that chosen global key points to.

All (or almost all) local keys exist in the form of XML file in the calibration database. The database is available on:

- GitLab
- In read-only mode on LxPlus under path: */cvmfs/na61.cern.ch/ReleasesDB/v3/Shine*

A given calibration will be loaded to reconstruction process only when it "covers" the data, i.e.:

- When reconstructed run number is within calibrated run range
- When time (timestamp) of reconstructed run is covered by the calibration (drift velocity calibration)

Reconstruction with custom calibration

Reconstruction with custom calibration requires modification in global key. DO THIS ONLY LOCALLY!

Every global key has its own *standardIdealDetConfig.xml* file that will have to be downloaded to local directory with several other files and modified before running reconstruction.

1. Download reconstruction setup files as in case of basic reconstruction.
2. Choose the global key which your custom calibration will be based on. Download its *standardIdealDetConfig.xml* file.
3. Edit *standardIdealDetConfig.xml*:
   - 3.1 Find the detector element which calibration you want to customize.
   - 3.2 Download its manager XML file and the files that the manager points at.
   - 3.3 Customize the calibration in these files according to your needs.
   - 3.4 In *standardIdealDetConfig.xml* change the path to point to your local customized manager.
4. Edit *bootstrap.xml*: Replace `standardIdealDetConfig` to point to your local *standardIdealDetConfig.xml*.
5. Download RAW file.
6. run *runModuleSeq.sh* with proper parameters (use original global key number).

Tips and hints

The example based on reconstruction of just one file. However, typical reconstruction is done for up to hundreds or thousands of files running in parallel. More optimization and automatization is needed. Below are some tips to run the jobs in more optimized way.

- Run reconstruction for one chunk per job – it is easier to control possible reconstruction errors and crashes
- Copy the input RAW file to the local directory of the job – it saves a lot of headaches with problems with data transfer between CASTOR and local directory
- When reconstruction is successful, copy the final SHOE file to EOS (using `eos cp`) and to CASTOR (using `xdrcp`).
- Write your reconstruction script is such way, that, if there was a problem with one stage of the reconstruction job, then cancel further stages of reconstruction – we do not want corrupted SHOE files among properly reconstructed ones.

Few links:

- The reconstruction scripts and setups that I use are available on GitLab: https://gitlab.cern.ch/bmaksiak/productions_shinew
- HTCondor Quick Start Guide: https://twiki.cern.ch/twiki/bin/view/NA61/HTCondorQuickStart
- HTCondor real-time job monitor: https://monit-grafana.cern.ch/

Common errors

Wrongly set env variables

Error

```
make: shine-offline-config: Command not found
make: shine-offline-config: Command not found
make: shine-offline-config: Command not found
...
runModuleSeq.sh: line 106: ShineOffline: command not found
```

Source: Environment variables were not set.

Solution: Set proper 32-bit environment variables.

Wrong variables set

Error:

```
[ERROR] Utilities/Reader/ReaderErrorReporter.cc:59: fatalError: Fatal Error at
"<SOMEPATH>/bootstrap.xml", line 21, column 31: unable to open external entity
'/cvmfs/na61.cern.ch/Releases/SHINE/v1r17p1/x86_64-centos7-gcc8-opt/
config/standardLegacyClientConfig.xml'

[ERROR] Utilities/Reader/Reader.cc:228: Parse: An error occurred during parsing
message: unable to open external entity '/cvmfs/na61.cern.ch/Releases/SHINE/
    v1r17p1/x86_64-centos7-gcc8-opt/config/standardLegacyClientConfig.xml'
terminate called after throwing an instance of 'utl::XMLParseException'
  what():  An error occurred during parsing, message: unable to open external
  entity '/cvmfs/na61.cern.ch/Releases/SHINE/v1r17p1/x86_64-centos7-gcc8-opt/
   config/standardLegacyClientConfig.xml'

runModuleSeq.sh: line 106: 29578 Aborted
(core dumped) ShineOffline -b $BOOTSTRAPFILE
```

Source: 64-bit variables were set instead of 32-bit

Solution: Log out and log in again to the terminal and set proper 32-bit variables. If this does not work, double check if you do not have setting of environment in your *.bashrc* file.

No global key

Example for a global key 15_044 which did not exist at the moment of creating this presentation

Error:

```
[ERROR] Utilities/Reader/ReaderErrorReporter.cc:59: fatalError: Fatal Error at
"<SOMEPATH>/bootstrap.xml", line 20, column 27: unable to open external entity
'/cvmfs/na61.cern.ch/ReleasesDB/v3/Shine/detConfig/15_044/
standardIdealDetConfig.xml'

[ERROR] Utilities/Reader/Reader.cc:228: Parse: An error occurred during parsing
message: unable to open external entity
'/cvmfs/na61.cern.ch/ReleasesDB/v3/Shine/detConfig/15_044/
standardIdealDetConfig.xml'
terminate called after throwing an instance of 'utl::XMLParseException'
  what():  An error occurred during parsing, message:
 unable to open external entity
 '/cvmfs/na61.cern.ch/ReleasesDB/v3/Shine/detConfig/15_044/
  standardIdealDetConfig.xml'
runModuleSeq.sh: line 106: 20520 Aborted (core dumped) ShineOffline -b $BOOTSTR
```

Source: Non existing global key (or database that is used does not contain chosen global key). Note that this error is very similar to the one from previous slide.

Solution: Double check if this is the key that you want to use and the database path is correct. If everything is correct, contact database maintainer.

No calibration for the run

Error:

```
[WARN] ManagerRegister.h:162: GenericGetData: None of the managers (<LONG LIST>
(...)
[INFO] Framework/Managers/Interface/VManager.cc:69: CanAnswer: Manager ...
cannot answer request typeIdName = 'det::Momentary<double>', component = 'TPC'
(...)
[ERROR] Framework/Managers/Interface/ManagerRegister.cc:148: NotFoundAndThrow:
Get data failure for typeIdName = 'det::Momentary<double>', component = 'TPC',
property = 'globalT0', componentIndex = { } for run 20455
and time 2015-02-22T06:59:13Z (UTC), 1108623569 (GPS), Unix:1424588353s
[ERROR] Modules/General/EventFileReaderSG/EventFileReader.cc:140: Process:
Get data failure for typeIdName = 'det::Momentary<double>', component = 'TPC',
property = 'globalT0', componentIndex = { } for run 20455
and time 2015-02-22T06:59:13Z (UTC), 1108623569 (GPS), Unix:1424588353s
```

Source: One of calibration values does not have coverage for a given run or timestamp. Notice the `det::Momentary` which is the core class of calibration values and is responsible for covering run range or period of time with a valid calibration value.

Solution: Double check if this is the run you want to reconstruct. Ask the database maintainer if the correct calibration values were put to database.

Shared memory problems

Error

```
(at some moment of the reconstruction)
DSBOOK Error: 110 Too many entries
Requested:                  3697 kB for point_mtl
.DSBOOK Error: 110 Too many entries
Requested:                  7390 kB for point_mtl
.DSBOOK Error: 110 Too many entries
(...)
(then, a long list of Shared Memory Segments)
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x0000001e 128942082  prodna61   666        3786736    0
0x0000001f 128942083  prodna61   666        3786736    0
0x00000020 128942084  prodna61   666        3786736    0
(...)
```

Source: DSPACK operates on "shared memory" which is a special kind of operational memory that can be shared between processes. The shared memory is just a small part of RAM. Several reconstruction processes running in parallel on one machine (e.g. HTCondor worker node) may exceed the shared memory and crash because of that.
Solution: Resubmit the reconstrucion job. If this still fails, then the worker node may have insufficient shared memory resources. Running the job interactively on LxPlus mostly helps.

See JIRA issue SHINE-207 for more details.

Problems with copying

There are various errors that happen when copying RAW file from CASTOR to the local directory (e.g. *Operation expired*) or when copying SHOE file to CASTOR or EOS.

Source: It just happens.

Solution:

- In case of copying RAW file from CASTOR, double check if the file is staged. If yes, just resubmit the job.
- In case of copying SHOE file to CASTOR and/or EOS, resubmit the job.
  But if file was copied properly to one place, but not copied to another, you may try to copy it manually without resubmission.

Frequently Asked Questions

Where is my production?

Where to find information about the production I would like to analyze?

- **From e-mails** sent to na61-soft with topic: *Mass production XXX ready*.
- **On TWiki** in section "List of productions"
  Choose year of production, then in table of contents choose the dataset and click the last entry which is the most recent production
- **For more experienced users:** find by yourself on CASTOR, EOS:
  - Prefix for **CASTOR**: */castor/cern.ch/na61/prod*
  - Prefix for **EOS**: */eos/experiment/na61/data/prod*
  - Dataset directories are arranged in logic:
    `<beam>_<target>_<energy>_<year>`, for example `Ar_Sc_150_15`
  - There may be multiple production directories inside dataset directories. Choose the one with the most recent global key (i.e. first number in the directory name) and suffix `_phys` or `_phys_<SOMETHING_MORE>`.

Every mass production has its **tag files** (text files with paths to all SHOE files ready to analyze). You may find it on AFS.

1. Navigate to */afs/cern.ch/na61/Calibration/prod*
2. Go to dataset directory (following the same logic as mentioned above)
3. Go to directory *tags*
4. Find the best tag file for you (there are four flavors: MegaSHOE files on CASTOR, MegaSHOE files on EOS, MiniSHOE files on CASTOR, MiniSHOE files on EOS, older productions have additionally DSPACK files on CASTOR).

What are our IT resources?

- **EOS disk space:** Currently, 330 TB for entire collaboration (CERNBox does not count here) – this is very limited disk space!
- **CASTOR disk space:** "infinite"
- **HTCondor resources:**
  - Entire NA61/SHINE experiment has nominal limit to run about 2000 jobs at the same time.
  - Within collaboration the quota is divided between normal user accounts (*wj* group) and production accounts (*wj-production* group) in ratio: 60% for *wj* and 40% for *wj-production*.
  - Production accounts (*prodna61* and *na61mc*) share quota equally 50/50.
  - The momentary quota fluctuates all the time which is a result of so-called *fair-share* approach introduced by CERN IT.

Useful CERN IT services

- TWiki – manuals, instructions, how-tos
- Indico – organizing and managing meetings
- CERNBox – analogy of Dropbox for CERN
- GitLab
- JIRA – reporting and tracking the progress of problems
- Overleaf – online and collaborative LaTeX editor
- ServiceNow (SNOW) – main CERN Service Desk
- OpenStack – building your own virtual machines
- Mattermost – real-time chat portal

How to work with CASTOR? Environment

CASTOR is the magnetic tape storage system. Most of data stored there cannot be accessed instantaneously. Data needs to be staged (copied from tapes to hard disk cache that can be accessed by user).

Access to CASTOR will work properly only after Shine environment will be loaded and additional environment variable *STAGE_SVCLASS* will be set!

```
SetShineEnv <shine_ver>/<32/64bit>
export STAGE_SVCCLASS=na61
```

How to work with CASTOR? Basic commands

nsls – it is CASTOR's analogy for *ls* (can be used without need of staging)

- **Main NA61/SHINE directory**: */castor/cern.ch/na61*
- **User directory**: */castor/cern.ch/user/%user_first_letter%/user/*

stager_qry – check if a given file is staged

- **Example**: `stager_qry -M /castor/cern.ch/na61/<path_to_your_file>`
- The output will be *STAGED*, *STAGEIN* (during staging) or *Not in this service class* (not staged).

stager_get – request a file to be staged

- **Example**: `stager_get -M /castor/cern.ch/na61/<path_to_your_file>`
- **Advice**: Request staging of many files at once – it is more optimal for staging machines
- Staging takes some time: average is about 4 hours

xrdcp – copy file from CASTOR to local directory (file has to be already staged)

- **Synopsis**:

```
xrdcp root://castorpublic.cern.ch/<full_path_to_file> <local_path> -OSsvcClass=na61
```

- **Example**:

```
xrdcp root://castorpublic.cern.ch//castor/cern.ch/na61/17/La/Xe150/run-032848x012.raw . -OSsvcClass=na61
```

- Mind prefix *root://castorpublic.cern.ch/*, double slashes // and option *-OSsvcClass=na61*

Vertices

What are: MainVertex, PrimaryVertex, BPD vertex, Fitted vertex? What are differences between them?

PrimaryVertex is the vertex that is considered as the vertex from which all particles produced in an event come from. It may have different sources:

- **BPD vertex (ePrimaryFitBPD):** when Z position of the vertex comes directly from where the target is (so, position on Z axis is fixed and only XY position is fitted). Used in reconstruction of data with thin target (Be, Sc, La, Pb).
- **Fitted vertex (ePrimaryFitZ):** when Z position is fitted as well. Used in reconstruction of data with long target (LHT, targets for neutrino runs).

ePrimaryFitBPD and *ePrimaryFitZ* are types of *PrimaryVertex*.

MainVertex is equal to one of those types.

- If reconstruction was done with -**pp** mode, **MainVertex** will be equal to **PrimaryVertex(ePrimaryFitZ)**
- If reconstruction was done with -**pA** mode, **MainVertex** will be equal to **PrimaryVertex(ePrimaryFitBPD)**

Vertices – more details

- Fitting of the vertex is done always – the structure of *PrimaryVertex(ePrimaryFitZ)* should always exist
- If *-pA* mode was done the *MainVertex* should be equal to *PrimaryVertex(ePrimaryFitBPD)*. But if the BPD fit fails, the *MainVertex* will be assigned to *PrimaryVertex(ePrimaryFitZ)*. However, the event will be most probably junk anyway.
- We DO NOT have *-AA* mode in the wrapped reconstruction chain. *-AA* mode is a relict of old legacy legacy reconstruction.

Further reading:

- *Legacy/Clients/R3D/src/vtx/vtxfit.F* for exact fitting procedure
- *Framework/Event/RecEvent/RecEventConst.h*, *VertexConst::EType* for types of vertices
- *Legacy/EventIO/NA49DST/NA49ToRecEvent.cc*, *GetVertexType* for how the conversion between iflag of vertex in legacy to vertex type in Shine

Thank you