

MINOS “Keep Up” Processing and Beam Data Acquisition

Robert Hatcher*¹, Arthur Kreymer¹, and Rashid Mehdiyev²

¹*Fermi National Accelerator Laboratory*

²*University of Texas, Austin*

August 5, 2013

1 Purpose

The MINOS “keep-up” processing stream uses the MINOS reconstruction and analysis code code to produce SNTP (standard ntuple) and DST (data summary tuples) files from the data in a timely “near offline” fashion. This processing performs separate passes over the input files for beam and cosmic triggers, but done as part of the same job unit (for bookkeeping and file handling purposes) for each file. The reasons for doing this processing include:

- The processed data uses slightly out-of-date calibration, but the result is used by the calibration group as input into the generation of up-to-date calibration constants. These constants are put into the database, after validation, for when a second pass of processing is done for “physics” purposes.
- Results from this pass serve to provide data quality feedback on the MINOS detector performance and used to identify problems that aren’t apparent from the immediate raw data (online monitoring).
- The resulting DST files with fully reconstructed physics quantities are used to monitor the beam performance in a manner not possible from the accelerator ACNet devices. For example, changes to the neutrino energy spectrum was a primary indication of problems with target degradation during the last run.
- The MINOS data is also used by MINERVA for track matching, as MINOS serves as their muon spectrometer and thus for them to do full reconstruction (i.e. their own “keep-up”) they must have these files.

It is desirable that the last three items get done with as little delay as feasible in order that they serve their purpose.

*rhatcher@fnal.gov

2 Data Sources

The data used to generate MINOS DST files comes from several sources:

- **DAQ:** this data stream is the readout of the detector proper; it includes triggers from both the beam (\$A9) and cosmic rays. Files (and run) boundaries are independent of any other sources.
- **DCS:** the “slow control” stream reports informations such as magnet status, HV, and other environmental statuses. These are used to determine if the detector was in a healthy state and whether the results are reliable.
- **SpillTiming:** is an independent system of keeping track of beam spills by recording the time of beam (\$A9) signals as seen by the MINOS near detector hardware; these timestamps are processed and put into the MINOS `spilltimend` database table for offline use. This system is also responsible for sending spill time information to the MINOS far detector for triggering.
- **Beam:** provides information about individual beam spills; primary quantities include “proton-on-target” information and values used to determine spill quality (e.g. targetting information). This data stream records all the available information and some of the device quantities are manipulated (e.g. fitting to reduce SWIC and BPM arrays down to a few values) before they are entered into the MINOS `beammonspill` database table.

Before the DAQ files are processed, MINOS requires that the other three sources have updated the database. In addition to checking that the relevant time period is covered, cross-checks are made to ensure that the `spilltimend` and `beammonspill` tables have recorded the same number of spills to within a specified tolerance.

2.1 Frequency and Latency

In the “low energy” MINOS-era NUMI running, “keep-up” processing was performed once a day, at approximately 11:30pm local time for the DAQ files that were at least 5 hours old. The desire for the future is to perform this processing three times daily. The higher intensity and higher energy of the beam means that plots that required 24 hours of data to get reasonable statistics could be made more frequently. The plan is to have the database tables filled covering the 8 hour periods starting at 00:00, 08:00, 16:00 UTC. How long after each of those periods one must wait before beginning processing is, at this time, probably determined by the beam data stream. The spill time stream will be updated hourly with a latency of no more than 15 minutes. The DCS frequency/latency is under investigation.

The MINOS understanding of acquiring the beam data is that normally the data will be available in the “historical” `ifbeam` database[1][3] within a few minutes after the actual spill. If one takes a conservative approach of not attempting to get spill information that happened in the previous 15 minutes, and that acquiring 8 hours of data and processing it into the `beammonspill` database table takes no more than 90+15 minutes, then “keep-up” processing for 00:00, 08:00, 16:00 UTC time periods should begin no earlier than 02:00, 10:00, 18:00 UTC respectively.

3 Beam Data Acquisition

Prior to the new `ifbeam` system (2005-2012), MINOS used the XML-RPC interface to get the device data from the accelerator division’s ACNet system. A list of devices was supplied and the XML-RPC system would return the data to the `bdp` process which would write the data to a `.mbeam` file. Each record contained all the available device information for a given trigger (\$A9). The `bdp` wrote a new `.mbeam` files for every 8 hours (00:00, 08:00, 16:00 UTC).

The `.mbeam` files were then processed through a standard MINOS job to summarize the data and make `beammonspill` database entries. This involves a variety of cross checks and data manipulation (fitting, etc) on a number of devices. Exactly which devices are necessary for this procedure is scattered amongst the code, though with some detailed code inspection and perhaps some empirical tests it is possible to determine a minimal device list for this processing. These two steps are shown in Figure 1.

The approach MINOS has taken, publicly discussed as early as 2013-01-17[2], is to continue to create `.mbeam` files of exactly the same format and leave the downstream portion of the processing unmodified. This left the experiment only to have to write a new front-end to the process that formats and writes the files.

3.1 Beam Data Size

There is no MINOS data storage rationale for trimming the device list from that of all possible devices. The total storage for beam data from roughly 500 devices covering the years 2005 through 2012 is less than 1 TB in total. The `.mbeam` files are compact in the structure and compressed as they are being written. The drawback is that data from these files that isn’t summarized into the `beammonspill` database table can only be extracted by using the MINOS C++ framework. Access to the data under that system isn’t hard and examples exist, but it is a potential barrier and can be inconvenient. The sum of the `beammonspill` summary data takes up roughly 12 GB in the database.

3.2 Beam Data Failure Modes

Near the end of the previous data taking run the XML-RPC system started suffering from a few failure modes. Some data was lost due to incorrect network traffic shaping or outages. Other data was partially lost when an individual device’s readback failed and the remained of the devices in the list would not proceed until the device timed out – this lead to some instances where most of every other spill’s information was lost.

In the XML-RPC system once data for a spill was not returned by the system, it was generally lost forever. MINOS ran a parallel secondary `bdp` process for redundancy, and on a few occasions some spills were recovered by extracting what was possible from the primary ACNet system.

The new `ifbeam` database approach limits these failures modes by making the the network administrators aware of this the critical traffic, having redundancy in the system, and using a system where if one device hangs up other device reads are unaffected.

But with these improvements comes some uncertainty about whether the accessible data is up-to-date. The experiment’s understanding is that this should, in normal operations, happen on a time scale of less than 15 minutes. But also that based on the `ifbeam` database itself there can be no indication that it is safe to read from it (no “high water mark”).

For keep-up processing a few missing spills (*acceptable tolerance%?*) is probably acceptable, larger losses (whole periods of 5-10 minutes or more) would probably warrant holding off keep-up processing until the data arrives. A cross check between `spilltimend` and `beammonspill` can't be used to avoid reading the `ifbeam` database, but it could hold off keep-up and force an investigation into the mismatch. A spill counting pass over the `ifbeam` database could be made for comparison with `spilltimend` before the pass to actually generate of the `.mbeam` file.

Until an alternative solution can be fielded, MINOS will probably run a second pass over the `ifbeam` database 3 or 4 days later and use that to look for cases of data arriving late. A check would be made that the two files have the same spills with the same number of devices read for each spill. If there were a discrepancy, presumably because the later file had additional data, then the new `.mbeam` file could replace the original and be reprocessed, otherwise it would be scrapped.

3.3 Beam Data Goals

[how to quantify any of this?]

- **performance:** the current system takes 10 minutes to acquire 1 hour's worth of data; while it is possible to run in that mode it seems that there should be possible optimizations that could greatly improve this.
- **latency:** caused by the sync of the "historical" `ifbeam` database is a fundamental feature of the design and can't be avoided. In normal operations this is probably acceptable, but it would be useful to have some firm number from the `ifbeam` database developers on what an appropriate estimate of the latency is.
- **lossage**
 - **delayed data:** this is data that is (initially) missed because it wasn't available when the `ifbeam` database was read. How to identify it? How to recover from it?
 - **lost data:** this is data that just never will appear. How to signal that it isn't simply delayed?

4 Extra Notes

4.1 MINERVA

MINOS currently supplies MINERVA with the digested beam information and they have no immediate plans for converting to the new system. Since MINOS "keep-up" processing will not proceed until that is available, MINERVA might be advised to now process any of their data that needs beam information until the MINOS files covering the same time period are available.

4.2 Times

The *time* associated with any datum or data is not necessarily obvious. This is diagrammed in Figure 2.

- **spill:** this should be the time of the actual spill, e.g. \$A9; such actual times should be recorded in the MINOS `spilltimend` database table independent of `ifbeam` database information.

- **device:** each device readback has an associated time; Phil Adamson’s nomenclature has this as the “DAE” (data acquisition event?) time.
- **record:** this is the time associated with a collection of ACNet data: the readout of multiple devices for the same spill. Ideally this would be the same as the spill time, but at a minimum it should serve as a proxy for the spill time and be close enough as to leave no ambiguity as to which spill it is associated with.

In the old XML-RPC scheme it was close to the time of the callback, and thus delayed by 500 msec from \$A9. The old code set it to the earliest “DAE” time of all the devices.

In the new `ifbeam` database scheme, when using `BeamFolder::GetTimeList()` it is some [criteria?] time associate with the collection. Allowed deviations from this time to be included in the collection is adjustable, but by default 50 msec. Empirically, with that criteria, scatter seems to be ± 42 msec.

- **content:** for some devices that return more than one value there is a time embedded in the content; this in PA’s nomenclature is the “VME” time.

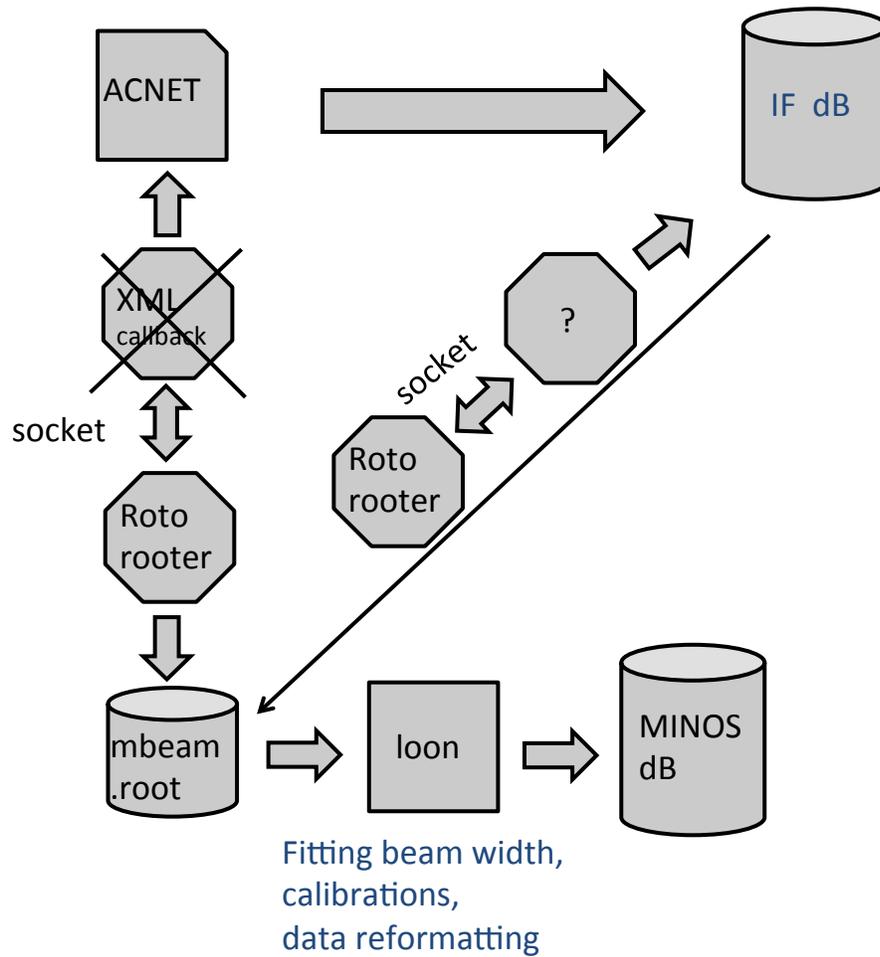
5 Beam Data Action Items

1. Method for getting definitive spill times (\$A9) for a period of time.
2. For any given time period what is the complete list of devices available to be retrieved.
3. Access to the data must be allow one to get all values for all devices for any given spill (i.e. there should be some bundle that contains all devices).
4. Efficient method of looping over spill times t_{rec} (or, until item 1 is satisfied, some proxy thereof) and getting all the data for that time. This must get the times in order and not split relevant spill data over more than one time.
5. Investigate SWIC time skew between `devtime` “DAE” time i.e. that associated with the device as an additional argument to `bf.GetNamedVector(trec,devname,&devtime);`, and the “VME” time embedded in the returned vector (for SWIC devices this is garnered from unscaled and combining elements 205-208).

References

- [1] A. Norman, et. al. IFBeamData redmine project page.
<https://cdcvns.fnal.gov/redmine/projects/ifbeamdata/wiki>.
- [2] MINOS Batch Group. MINOS Beam Processing redmine project page.
<https://cdcvns.fnal.gov/redmine/projects/batch/wiki/MinosBeam>.
- [3] I. Mandrichenko, et. al. Implementation of Beam Conditions Database for Intensity Frontier Experiments.
<https://cd-docdb.fnal.gov:440/cgi-bin/ShowDocument?docid=4478>.

Figure 1: MINOS action plan for adjusting to lost of XML-RPC protocol in acquiring the beam data as described in Section 3 as publicly presented at 2013-01-17 meeting[2]. Diagram by Rashid Mehdiyev based on white board drawings of Robert Hatcher.



Based on
¹R.Hatcher's scheme

Figure 2: MINOS XML-RPC time line, see Section 3.

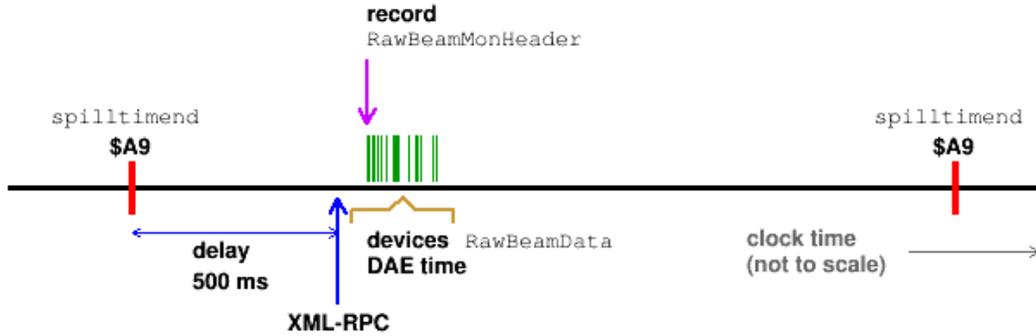


Figure 3: MINOS .mbeam record structure as written by the bdp process.

