

# MINOS Beam Data Acquisition and Processing Plan in the Post XML-RPC Era

Robert Hatcher\*<sup>1</sup>, Arthur Kreymer<sup>1</sup>, and Rashid Mehdiyev<sup>2</sup>

<sup>1</sup>*Fermi National Accelerator Laboratory*

<sup>2</sup>*University of Texas, Austin*

MINOS-docdb-10062-v3

October 9, 2013

## 1 Purpose

The success of the MINOS experiment relies in part on data acquired from the accelerator complex concerning the NUMI beamline. One of the key elements of this data is the protons-on-target associated with each spill. Other components represent measures of the quality of the spills from whence one can decide whether an individual spill was incorrectly targeted or otherwise abnormal and thus rejected.

This document is designed to layout the plan for acquiring that data and making it available for use in the reconstruction and analysis phases[3][4].

## 2 Overall Beam Data Plan

In the previous years of running (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+500ms` delay). The `bdp` collected this information, packed it and sent it to the `rotorooter` which wrote a `.mbeam` file. The `bdp` processor initiated a new file every 8 hours (on the 00:00, 08:00, and 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. These two steps are shown in Figure 1.

With the demise of the XML-RPC interface MINOS was required to change how this is done. Plans for how to react to this were publicly discussed as early as 2013-01-17[2], with the idea 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.

---

\*rhatcher@fnal.gov

The initial plan for the post-XML-RPC era was to use the `ifbeam` database to acquire the data to be sent to the `rotorooter`. After much effort this has proved to be impractical; it wasn't designed for what this approach would need, nor, when coerced to approximate it, did it have adequate performance.

The current plan is to receive the same *collector files* as are used for input to the `ifbeam` database; parse those and write MINOS standard `.mbeam` files.

### 3 Beam Data *collector files*

#### 3.1 File Format

The *collector files* are plain text files of a simple format, or a `gzip` compressed version of same. The plan is to make no assumptions about the filename relative to the contents of the file, though MINOS was assured that uniqueness of the filename could be assumed. The contents of the file consists of one or more *event lines*, each of which are followed by an unspecified number of *device lines*. Different collections of devices are assembled into separate files, and a redundancy of collector processors (on separate nodes that aren't co-located) means that there is generally duplication amongst the files. This redundancy and separation needs to be resolved before data is put into the MINOS `.mbeam` files. Files are assumed to never have data covering more than 24 hours; errant timestamp that break this rule will have to be dealt with by special means.

The *event lines* have start with a “\*” character, followed by the trigger string, a space character, and the event *timestamp*, ala:

```
*e,a9,e,500 1373570364033
```

which is a “e,a9,e,500” trigger at 2013-07-11 19:19:24.033 UTC. The form of the event label is a set of comma separated fields, the second is the base trigger (MINOS timeline is trigger relative to \$A9) and the fourth (if present) is a delay after that, in milliseconds, where the actual data acquisition is to take place. The general reason for such a delay is to allow the the front-end to finish digitizing and formatting the data[4].

The *device lines* are of the form

```
<device name> <timestamp> <units> <scalar value> <array values>
```

where fields are separated by the *tab* character. Examples look like:

```
E:MM2HV2    1373570604609 Volt -296.90625 null
E:MMA3DS[] 1373570364033      null      {0.0,0.0,0...}
```

The “[ ]” in the device name indicate that the device is multi-valued and the fifth field holds the values (enclosed in braces (“{ }”) and comma separated); otherwise there is only the single scalar value in the fourth field. The units field might be blank (null string). The timestamp for this device's entry needn't be exactly the same as the *event line* that it is associated with. In MINOS terminology the device time is referred to as the DAE time.

### 3.1.1 File Format Modification 2013-10-08

A proposal was floated circa 2013-09-26(ish) to modify the contents of the collector files. Events will no longer be labelled as “e,a9,e,500” for triggers 500ms after the \$A9 event and the times will no longer be the DAE times. Instead the *event lines* starts with a “\*” character, followed by the *base* trigger string, a space character, the event timestamp, a space character, and the delay in milliseconds, ala:

```
*e,a9 1373570363533 500
```

which is data collected 500ms after a \$A9 trigger with the time adjusted back to approximate the actual \$A9 time. Similarly, the times associated with device data will no longer be the actual DAE times, but the approximate \$A9 time.

MINOS code was modified on 2013-10-08 to handle this new format and revert the times to DAE times. This allows downstream code (which already has a means of managing these offsets) to continue without change.

## 3.2 Collector File Creation

The Computing Sector DA Group<sup>1</sup> will be responsible for creating the files. These are the same files that they will use to fill the “historic” (vs. “real-time”) `ifbeam` database.

For (non-definitive) reference here is an overview of how this happens. A collector process is started for a given combination of *event label* (e.g. `e,a9,e,500`) and a list of devices. Such a combination is called a *bundle*. For redundancy purposes, there are generally two collectors running for every such bundle, the second started at an offset time so that in the case of a restart there are no gaps. The bundles interesting to MINOS are `NuMI_Physics` (with a *long* retention period, i.e. forever) and `NuMI_Monitoring` (with a *short* retention period, approximately 1 month). This means that for any particular spill there are four files that could potentially have data for that spill.

The collector process makes a request for data, based on the combination of the bundle, from an accelerator division *broker* process. The broker process sends data asynchronously to the collector process. A collector file starts with an *event line*, that is triggered by the first device data line it receives from the *broker* process. The *event line* timestamp is that of the device it saw. It then collects device lines until it sees one with a timestamp more than 50ms away (asymmetric?), upon which it starts a new *event line*. If the collector system sees a device line earlier than the time on the *event line* it will not adjust the time associated with the event. This means network packet ordering might change *event line* times if different collectors see the devices in a different order as devices have a jitter in their DAE time.

Files are closed out by the collector process after  $N$  events ( $N \sim 100$ ) or a pre-specified timeout period (5 min?).

---

<sup>1</sup>Computing Sector/Scientific Computing Services/Scientific Data Processing/Databases and Applications Group

### 3.3 Acquisition

After the files are closed the DA group will process them into the “historic” database, *and* send a copy to MINOS. Whatever scheme is used for copying files must satisfy four properties:

- ownership `mindata:e875`
- hidden while in transit, not visible if failed to copy completely
- if MINOS removes a file from the destination directory, a new version should never be re-copied from the source.
- if MINOS falls behind in removing copies, a file will never be removed from the destination directory even if it is removed from the source area.

Such a system would use `scp` with a special kerberos service principal or `rsync` to copy files to the designated location, `/minos/app/BEAMDATA_CF/IMPORT`, and owned by the `mindata:e875` account. The DAG group has chosen to push the files separated into two directories `short` and `long`, presumably based on their own retention policy. No particular assumption will be made by MINOS to depend on this, but it will be carried over to the reorganization scheme as `subdir`, but MINOS will retain this information in the event that it becomes useful to MINOS at some later time.

Files are copied to that location using a temporary name such as `basename.in_transit` or otherwise hidden (i.e. a name starting with `.` as `rsync` will do). After the copy is successfully completed, a second step would rename them to the proper name (i.e. remove the `.in_transit` extension). This prevents downstream processing from accidentally picking up files that aren’t yet completely copied (either actually in transit or for an interrupted copy). Collector files can be input into the MINOS system either as straight text files or already `gzip`’ed.

### 3.4 Organization of Individual Files

Because the files contain *event triggers* for only a short period (typically a few minutes) they tend to be quite small and there are lots of them. The small size of the files argues for these to be kept on the `/minos/app` disk rather than the `/minos/data` disk; the later has a 32K blocksize and is thus unsuitable for collections of numerous small files. So as not to overwhelm the filesystem it is advisable to keep the files in directories with no more than a few thousand files at most.

The plan for MINOS is to categorize the files based on the event triggers in the file, both trigger label and delay (e.g. “`e,a9,e,500`” or “`e,a9+500`”) and earliest timestamp. This will also facilitate processing the files by making it easier to decide which files to use as input (which can’t be decided on from the file name itself).

The chosen scheme is to move files into a hierarchical structure of the form:

```
/minos/app/BEAMDATA_CF/ARCHIVE/<trig>/<year>/<month>/<day>/<subdir>
```

where `<trig>` will be a string such as `e,a9,e,500`. The `<year>`, `<month>`, `<day>` are numeric values for UTC time of the first trigger and `<subdir>` is the original subdirectory (e.g. `short` or `long`) from whence the file was imported. *After the 2013-10-08 collector file change:* files with the third field on the event lines, the `<trig>` string is formatted as `e,a9+500`.

Periodically the `IMPORT` area is scanned for files without the `.in_transit` extension and older than some set time (5 minutes) by the `catalogue_cf` script. Any files found are classified and moved

to the ARCHIVE hierarchy. At the top of the ARCHIVE file tree there is a file AAA\_FILE.CATALOGUE with additional information, such as last timestamp and number of lines and triggers, that have been moved into the tree<sup>2</sup>.

Files moved into the ARCHIVE should also be compressed. A sampling of existing collector files shows that while very, very small files could be enlarged, in general there is a significant reduction to be had (1859 MB became 82 MB). The exact gain might vary with the mix of triggers, the composition of devices and the repeativeness of their values.

### 3.5 Processing into .mbeam files

The `cf2rr` executable and the `cf2root` script that drives it (source found under the MINOS BeamData package, in the `bdcf` subdirectory) are responsible for creating a `.mbeam` file. These files are named `Byymmdd_hhmmss.mbeam.root` where the time represents the UTC time of the start of the window (not necessarily the time of the first record) covered by the file. MINOS will continue to create a new file for every every eight hours on the 00, 08 and 16 UTC.

```
Usage: cf2root [options] B[YY][MM][DD]_[hh][mm][ss]
       where BYYMMDD_hhmmss encodes the desired start (UTC) time
```

```
-h --help           print this help and quit
-m --mailto=LIST   who to inform on error (unset => std out)
                   [rhatcher@fnal.gov]
-t --trigger=TRIG  trigger string [e,a9,e,*:e,a9+[1-9]*]
                   (colon separated or multiple flags)
-d --duration=DUR  duration of .mbeam file [8h]
-e --epsilon=EPS   event merging epsilon (ms)
-W --mergew=MW     max possible window (sec) for merging
-M --mindev=MDEV   minimum # devices/record
-s --subdir        e.g. "short long" [*]
-r --remake        allow .mbeam file to be re-made
-b --base=BPATH    alt base path [/minos/app/users/BEAMDATA_CF]
-i --indir=INSUB   alt input area [ARCHIVE]
-o --outdir=OUTSUB alt output area [MBEAM]
-c --catfile=CFILE alt catalogue file [AAA_MBEAM.CATALOGUE]
-L --longcf=DELTA  longest duration for collector file [8h]
-f --force         force, even if another instance might be running
-O --write-empty   allow empty files to be created
-v                add to verbosity
```

The `cf2root` script is normally given a desired base for the file name (`Byymmdd_hhmmss`). By default the duration is assumed to be 8 hours (configurable from `getopt` flag). It must determine

---

<sup>2</sup>Any files that contain timestamps covering greater than 24h need special treatment. So far these look to be cases where the first timestamp is crazy old e.g. 1970-11-12 and subsequent times are reasonable 2013-07-xx. Current processing of these files treat any instances of initial timestamps before 2011-01 as bogus and continue looking in the file for later dates. If one is found it is used instead for classification purposes. If a case did arise where a file spanned more than 24 hours then a copy or link should probably be put in additional directories.

the list of collector files to pass to cf2rr; start the rotorooter process; start cf2rr; stop the rotorooter; and make a flag file to signal the archiver process that the file exists. As a byproduct it also produces a catalogue of files it has created and keeps the logs from cf2rr.

Usage: cf2rr [OPTIONS] collector-file-path(s)

Read data from IFBeam DB, reformat and send to MINOS beam rotorooter.

Mandatory arguments to long options are mandatory for short options.

-h, --help display this help and exit  
-t, --timestamp=TIME starting time in seconds since epoch  
-T, --timestring='YYYY-MM-DD hh:mm:ss'  
-d, --duration=VAL file duration (default 8hr, default units hours)  
use VALs or VALm for sec, min; e.g. 3600s  
-r, --rotonode=MACHINE node where rotorooter is running  
-C, --print-config print configuration and exit  
-v, --verbose increase verbosity  
-c, --count-spills simply count spills w/ no rotorooter  
-e, --epsilon=EPS merge epsilon (ms)  
-W, --merge=MW merge window (sec)  
-M, --mindev=MDEV merge records w/ fewer than MDEV devices  
-q, --dt-tolerance=MS complain about spill/device time diff  
-P, --print-opts=POPTS what extra to print  
-p, --pattern=INFILE pattern for file matching [\*]  
this -p flag can be used multiple times  
or use patterns separated by colons  
-0, --write-empty write file even if no records  
--bypass-roto continue after rotorooter connection/file errors

Use of positional arguments is a deprecated feature, use flags instead.

PrintOpts POPTS is a comma separated string of things to print

triggers = unpacking start of event records  
devices = unpacking values for devices  
files = files to be read (after wildcard expansion)  
missing = device only if it was missing from some spills  
dt = excursion extremes between spill and device time (ms)  
spillcnt = # of spills

cf2rr: current configuration:

Verbose 0  
TimeStampStart 0 (ms) 1970-01-01 00:00:00 (UTC)  
1969-12-31 18:00:00 (local)  
TimeStampStop 0 (ms) 1970-01-01 00:00:00 (UTC)  
1969-12-31 18:00:00 (local)  
Duration 28800000 (ms)  
RotoNode "minos-nearline.fnal.gov"  
Epsilon 200 ms MergeWindow 600000 ms MinDevices 15 DtTolerance 500

The `cf2rr` program is told a starting time and a duration and given a list of collector files (potentially a wildcard list). It must read the files; interpret the text into pseudo-records, sorting and merging duplicate data (parallel collector files for each bundle) and possibly merging nearby records (due to jitter in the *event line*); send the data to the `rotorooter`; and print some statistics.

## 4 Latency and Data Retention

On the `ifbeam` database side, data in the “real-time” db is only retained for an hour and potentially subject to loss depending on loads, but has a latency of a few seconds from the actual spill. The “historic” data has a longer latency (no less than the duration of the collector files) and data is either retained forever (“Physics” bundle) or a month (“Monitoring” bundle).

MINOS policy will be to process the 8 hour periods of collector files at 00, 08, 16 hour UTC (with overlap on both ends to allow for event coalescence). The resulting `.mbeam.root` files will have indefinite retention as they have always had. Additionally, the plan is, after sufficient time, to tar up older collector files and put those on tape as well.

Early indications are that the total size of the `gzip`'ed collector files is roughly 38.3 GB for a month's worth of `$A9` related files, 10 GB for `$8F` files, and a bit over 100 MB for the remaining sundry.

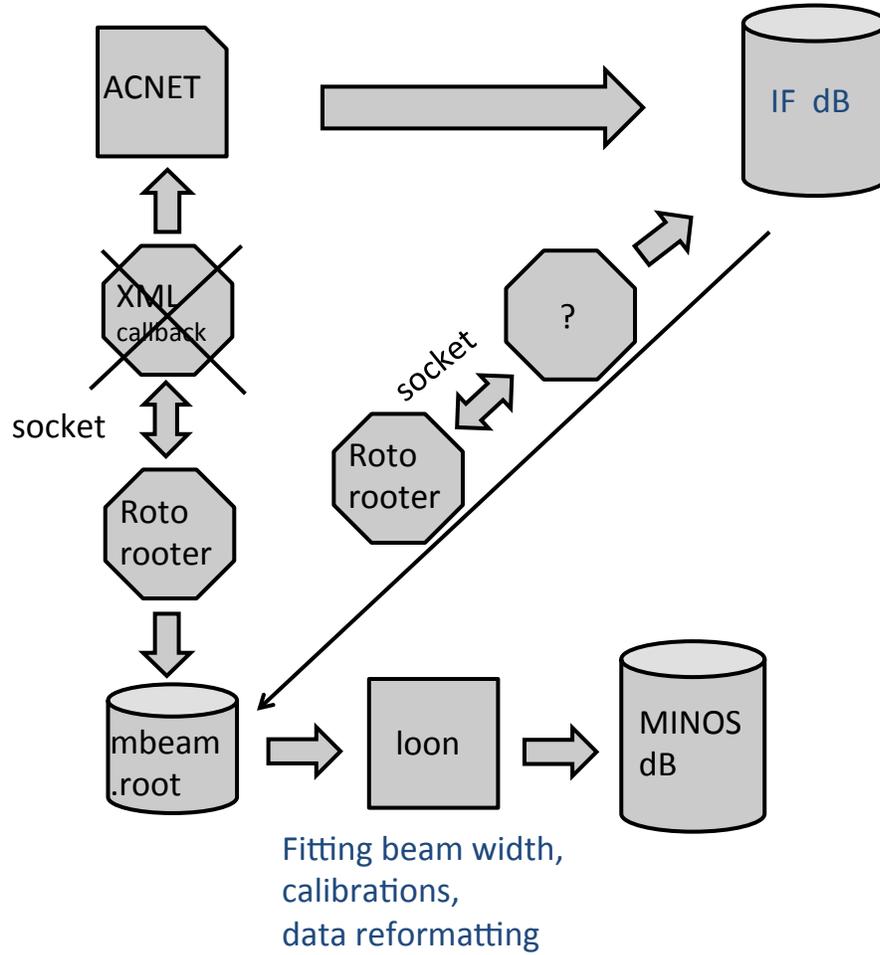
## 5 Terminology

- **collector file:** text files generated by the SC/DA Group containing beamline device readout information, both event triggers and the device data associated with them.
- **event trigger:** a condition which triggers the collection of device data.
- **device data:** a device, a timestamp (,units) and values.
- **timestamp:** times in the *collector files* are milliseconds since epoch 1970-01-01 00:00:00 UTC (note these are too large for a 32-bit integer).
- **bdp:** the original MINOS executable for interfacing with the XML-RPC system.
- **ifbeam database:** a database of beamline device data accessible from a Web-based interface[5] and C++ interface[1].
- **rotorooter:** a process that reads, from a TCP/IP socket, a binary representation of packed chunks of the MINOS `RawData` blocks; converts them into objects of the corresponding derived class; and writes them to a ROOT format file.

## References

- [1] A. Norman, et. al. IFBeamData redmine project page.  
<https://cdcvs.fnal.gov/redmine/projects/ifbeamdata/wiki>.
- [2] MINOS Batch Group. MINOS Beam Processing redmine project page.  
<https://cdcvs.fnal.gov/redmine/projects/batch/wiki/MinosBeam>.
- [3] Robert Hatcher et. al. MINOS Keep-Up Processing and Beam Data Acquisition.  
<http://minos-docdb.fnal.gov/cgi-bin/ShowDocument?docid=10051>.
- [4] Robert Hatcher et. al. Time Skew in SWIC Timestamps as Seen by the MINOS and ifbeam Beam Data Systems. <http://minos-docdb.fnal.gov/cgi-bin/ShowDocument?docid=10052>.
- [5] 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 2 as publicly presented at 2013-01-17 meeting[2]. Diagram by Rashid Mehdiyev based on white board drawings of Robert Hatcher.



Based on  
<sup>1</sup>  
R.Hatcher's scheme

Figure 2: MINOS XML-RPC time line, see Section 3.2.

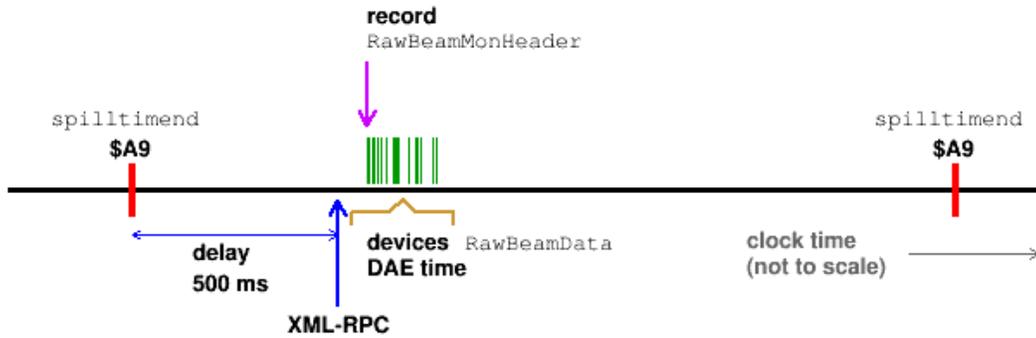


Figure 3: MINOS .mbeam record structure as written by the bdp process.

