

Time Skew in SWIC Timestamps as Seen by the MINOS and `ifbeam` Beam Data Systems

Robert Hatcher*¹, Arthur Kreymer¹, and Rashid Mehdiyev²

¹*Fermi National Accelerator Laboratory*

²*University of Texas, Austin*

August 6, 2013

1 Problem Report

When running the standard MINOS `.mbeam` files derived from the `ifbeam` db system to fill the MINOS `beammonspill` db table there were numerous reports of an unacceptable discrepancy between the DAE time of the `RawBeamData` sub-entry of the `RawBeamMonBlock` and the VME time that is encoded into the SWIC device’s data itself (Figure 3).

When the same processing is run on the original MINOS `.mbeam` file, generated at the time via the XML-RPC interface, discrepancies are identified in roughly 100 of 12000 spills.

2 Beam Data Acquisition Procedure

Both the `xml-rpc` and `ifbeam` files pack collections of device data into `RawBeamMonBlock` object, held by a `RawRecord`. The record has a timestamp that serves as a proxy for the spill time; this time is close to the actual time of the spill, but needn’t be exactly the spill time.

2.1 `xml-rpc`

In the `xml-rpc` approach data was taken in real-time as spills occurred. Following a delay (500 ms) after a spill the `xml-rpc` system collected data and returned a full set of device information to the `bdp` (beam data process) as a set of devices, with a “DAE” time and one or more values for each device. This was parsed; the earliest DAE time was used as the record time; and the data was packed into the output record. A schematic of the time line is shown in Figure 1.

2.2 `ifbeam`

The basic approach for reading the `ifbeam` db and filling records is shown in Section 5. The `BeamFolder` class is initialized with a starting time and “bundle” name. The `BeamFolder` is queried for a list of times and devices. That list of times represents some times at which devices have data; by default times within 50 msec are coalesced into a single collection. These times, as returned by `GetTimeList()`, serve as a proxy for the spill time and are used as the record time. The `BeamFolder` is then queried for all the device data associated with that record time. This is packed and written as records in the output file. When each device is queried for its data the timestamp associated with that record is also queried and used as the “DAE” time (ala. `dae` in `GetNamedData(trec,devname.plus_at,&value,&dae)`; and `GetNamedVector(trec,devname,&dae)`).

*rhatcher@fnal.gov

2.3 Extracted Data

Given a record time, as defined above, the MINOS `spilltimend` table was queried for the nearest spill.

To cross check that the data was for the same spill the toriod device was queried for its value and associated DAE time. These are consistent between the two systems when the 500 msec delay is accounted for in the `xml-rpc` system.

For the chosen SWIC device the DAE time was extracted. The 216 values encapsulate the raw wire data and a “VME” time, among other things. The VME time was extracted and is reported. To summarize the raw wire data a simple sum is made of all the relevant values in a manner similar to a checksum. This identifies the wire information to verify that the data hasn’t become disconnected from the VME time.

2.4 Test Results

The test example is for data starting on 2012-04-14 16:00:00 (1334419200 sec since epoch); all times are seconds relative to this. The original MINOS XML-RPC file started at 16:00:01, while the `ifbeam` db file collected values from 16:00:00. There are no actual spill that fills that first second, so that isn’t the issue.

Information was extracted from the first three records in each file.

	t_{spill}	t_{rec}	E:TRTGTD		E:M121DS		
			t_{DAE}	value	t_{DAE}	t_{VME}	$\sum_{i=104}^{199} r_i$
<code>ifbeam</code>	01.285083169	01.283999919	01.285	36.4279	01.283	59.267149320	0.677206
<code>xml-rpc</code>	01.285083169	01.760	01.785	36.4279	01.784	01.342443480	0.713218
<code>ifbeam</code>	03.351759458	03.351000070	03.351	36.151	03.351	01.325524680	0.713218
<code>xml-rpc</code>	03.351759458	03.833	03.852	36.151	03.851	03.389788120	0.714743
<code>ifbeam</code>	05.418491771	05.417999982	05.417	36.1057	05.417	03.401934680	0.714743
<code>xml-rpc</code>	05.418491771	05.900	05.918	36.1057	05.918	05.449747360	0.704672

For each of the three spills the `ifbeam` record time is fractionally earlier than the recorded spill time. While the DAE times for both devices are consistent with the spill time and the toriod values match, the SWIC time and raw sum appear to be off-by-one. The `ifbeam` system appears to be associating the previous spill’s SWIC data with a DAE time for a later spill.

3 Comparison with Direct Web Fetch

An attempt was made to see if the web interface gave a different result from the `BeamFolder` C++ interface. The first check was for the toriod information, picking out the three spills via:

```
curl -sL "http://ifb-data.fnal.gov:8089/ifbeam/data/data?e=e,a9&\
v=E:TRTGTD&t0=1334419200s&t1=1334419206s&f=csv&tz=&action=Show+variable"
```

```
Event,Variable,Clock,Units,Value(s)
"e,a9",E:TRTGTD,1334419201285,E12,36.4279252228
"e,a9",E:TRTGTD,1334419203352,E12,36.1510109169
"e,a9",E:TRTGTD,1334419205418,E12,36.1057293323
```

Substituting `E:M121DS\[\]` (extra slashes to protect square brackets from the shell) for `E:TRTGTD` returns the desired data or one can use a bundle and select the returned CSV line for `E:M121DS`. Both results were consistent with the `BeamFolder` approach. A spot check of value for indices 205-208 (211-214 or 209-212 when adjusting for the extra info and cut using 1-based indexing) gave values consistent with those of the C++ extraction from the `ifbeam` file. The query was:

```
curl -sI "http://ifb-data.fnal.gov:8089/ifbeam/data/data?\  
e=e,a9&v=E:M121DS\[ \]&t0=1334419200s&t1=1334419206s&f=csv&tz=" | \  
cut -d, -f4,211-214
```

```
curl -sI "http://ifb-data.fnal.gov:8089/ifbeam/data/data?b=NuMI_all&\  
t0=1334419200s&t1=1334419206s&f=csv&tz=" | grep "E:M121DS" | cut -d, -f1,209-212
```

```
1334419201284,6.21387371441,-7.57866145817,1.24393444624,7.50267036958  
1334419203351,6.21387371441,-7.57805108798,1.51585436567,2.24860377819  
1334419205418,6.21387371441,-7.5774407178,1.87170018616,0.730002746666
```

3.1 Times

The *time* associated with any datum or data is not necessarily obvious. This is diagrammed in Figure 1.

- **spill**: this should be the time of the actual spill, e.g. \$A9; such actual times should be recorded in the MINOS `spilltimend` db table independent of `ifbeam` db 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` db 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 embedded in the content a time; this in PA’s nomenclature is the “VME” time.

4 Inspecting the .mbeam files

To dump “interesting” things about the first 3 records in the .mbeam file use the MINOS offline code command:

```
loon -bq -r 3 dumpswic.C+ B120414_160000.mbeam.root
```

4.1 dumpswic.C

```
1  //#####
2  //# dump selected SWIC data from .mbeam file; code must be compiled
3  //# usage: loon -bq dumpswic.C+ <BYMMDD_hhmmss.mbeam.root>
4  //# rhatcher@fnal.gov
5  //#####
6
7  #include <iostream>
8  #include <iomanip>
9  #include <algorithm>
10 using namespace std;
11
12 #include "TSystem.h"
13
14 #if !defined(__CINT__) || defined(__MAKECINT__)
15 #include "JobControl/JobC.h"
16 #include "JobControl/JobCEnv.h"
17 #include "MinosObjectMap/MomNavigator.h"
18 #include "RawData/RawRecord.h"
19 #include "RawData/RawDaqHeader.h"
20 #include "RawData/RawBeamMonHeader.h"
21 #include "RawData/RawBeamMonBlock.h"
22 #include "RawData/RawBeamData.h"
23 #include "RawData/RawBeamSwicData.h"
24 #include "RawData/RawBeamPosData.h"
25 #include "BeamDataUtil/BDDDevices.h"
26 #include "SpillTiming/SpillTimeFinder.h"
27 #endif
28
29 void dumpRaw(const RawRecord* rawRec, bool verbose=false)
30 {
31     if (!rawRec) return;
32     cout << "----- RawRecord -----" << endl;
33     //rawRec->GetRawHeader()->FormatToOStream(cout,""); cout << endl;
34     //rawRec->Print("l"); // print list of raw blocks
35
36     VldContext vldc = rawRec->GetHeader()->GetVldContext();
37     cout << "VLDC " << vldc << " sec " << vldc.GetTimeStamp().GetSec() << endl;
38
39     // looking for RawBeamMonBlock
40     const RawBeamMonBlock* rbmb =
41         dynamic_cast<const RawBeamMonBlock*>(rawRec->FindRawBlock("RawBeamMonBlock"));
42     if ( ! rbmb ) return;
43
44     // 32-bit ROOT on my laptop is currently broken ... won't run this code
45     // if used in conjunction w/ Demo module ... so stop using Demo
46 #ifndef XYZZY__DARWIN_UNIX03
47     // from the raw record time we can find the nearest beam spill time
48     // get the header, so we can determine the record's timestamp
```

```

49     VldTimeStamp spill_ts =
50         SpillTimeFinder::Instance().GetTimeOfNearestSpill(vldc);
51     cout << "SpillTimeND " << spill_ts.AsString("c")
52     << " diff " << (vldc.GetTimeStamp()-spill_ts).GetSeconds()
53     << endl;
54 #endif
55
56     if (verbose) rbmb->Print();
57     cout << "RawBeamMonBlock TclkTrigger Event " << rbmb->TclkTriggerEvent()
58     << " Delay " << rbmb->TclkTriggerDelay() << endl;
59
60     std::vector<std::string> devlist;
61     devlist.push_back("E:TRTGTD");
62 #ifndef ALL_SWICS
63     std::vector<std::string> swics = BDDevices::SwicDevices();
64     devlist.insert(devlist.end(), swics.begin(), swics.end());
65 #else
66     devlist.push_back("E:M121DS");
67 #endif
68
69     for (size_t idev = 0; idev < devlist.size(); ++idev ) {
70         // loop over devices in our list
71         std::string devname = devlist[idev];
72         const RawBeamData* rbd = (*rbmb)[devname]; // get the sub-block
73         if ( ! rbd ) { cout << "no " << devname << " ?!" << endl; continue; }
74
75         // size_t dl = rbd->GetDataLength();
76         VldTimeStamp tdae(rbd->GetSeconds(),rbd->GetMsecs()*1000000);
77         cout << devname << " dae " << tdae.AsString("c")
78         << " [0]=" << rbd->GetData()[0] << endl; // print first datum
79         if ( idev != 0 ) { // at present only first device isn't a SWIC
80             RawBeamSwicData swic(*rbd);
81             static const int CALCULATION_OFFSET = 0;
82             static const int CALCULATION_BLOCK_COUNT = 8;
83             static const int CALCULATION_BLOCK_LENGTH = 13;
84             static const int RAW_OFFSET =
85                 CALCULATION_BLOCK_LENGTH * CALCULATION_BLOCK_COUNT +
86                 CALCULATION_OFFSET; // this is = 104
87             static const int RAW_LENGTH = 96;
88             // calculate the sum of the "raw" data ... just a verification
89             // that we're talking about the same info
90             double sum = 0;
91             for (int indx=0; indx < RAW_LENGTH; ++indx)
92                 sum += rbd->GetData()[RAW_OFFSET+indx];
93
94             VldTimeStamp tvme(swic.VmeSeconds(),swic.VmeNanoseconds());
95             cout << "         vme " << tvme.AsString("c") << " "
96             << "sum[" << RAW_OFFSET << ":" << RAW_OFFSET+RAW_LENGTH
97             << "]= " << sum << endl;
98
99             cout << "         [205:208]= "
100             << rbd->GetData()[205] << " " << rbd->GetData()[206] << " "
101             << rbd->GetData()[207] << " " << rbd->GetData()[208] << " "
102             << endl;
103         }
104     }

```

```

105 }
106
107 void myAna(const MomNavigator* mom) {
108
109     cout << "\n===== Record Set =====" << endl;
110
111     const TObject* obj = 0;
112     for (int i=0; (obj=mom->At(i)); ++i) {
113
114         cout << "[" << i << "] ";
115         if (!obj) cout << "empty slot" << endl;
116         else     cout << "Class: '" << obj->ClassName()
117                 << "' Name: '" << obj->GetName() << "'" << endl;
118
119         const RawRecord* rawRec = dynamic_cast<const RawRecord*>(obj);
120         if ( rawRec ) { dumpRaw(rawRec,false); cout << endl; }
121     }
122 }
123
124 void dumpswic(int nrec=9999999) {
125
126     cout << "processing beam file: " << JobCEnv::Instance().GetFileName(0) << endl;
127
128     JobC jc;
129
130     // create a path, Input::Get is implicit
131     jc.Path.Create("Spin", "RootCommand::Ana");
132
133     jc.Path("Spin").Mod("RootCommand").Cmd("AddLine/Ana      myAna(mom);");
134
135     jc.Input.Set("Format=input");
136     jc.Input.Set("Streams=BeamMon");
137
138     // for the dump of the block to have the raw values in hex format
139     // as well as the fully formatted decoded interpretation
140     // RawDataBlock::SetForceHexDump(true); //DebugFlags(0xffffffff);
141
142     jc.Msg.SetLevel("Dbi", "Warning");
143     jc.Msg.SetLevel("SpillTiming", "Warning");
144
145     jc.Path("Spin").Run(nrec);
146     jc.Path.Report();
147
148 }

```

Figure 1: MINOS XML-RPC time line, see Section 2.

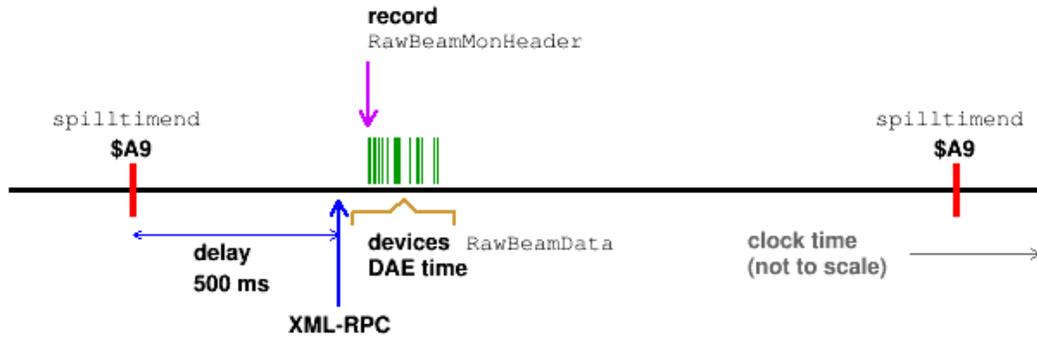


Figure 2: MINOS .mbeam record structure as written by the bdp process.

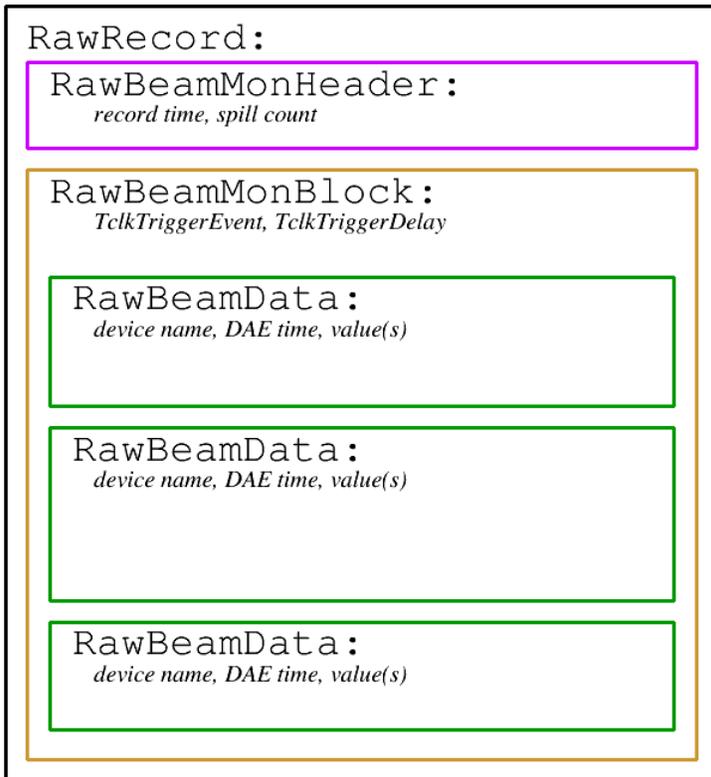


Figure 3: Relevant functions for unpacking the SWIC VME timestamps, from circa line 110 of `RawData/RawBeamSwicData.cxx`. Data comes as floats and needs to be unscaled and combined back to integers before the values can be used as second and nanoseconds since epoch.

```

1 // Timestamp block
2 // Only the GPS timestamp is relevant for non-miniBoone SWICs
3 static const int EVENT8FTIMESTAMP_OFFSET = 201;
4 // The first 2 words are combined to form a 32 bit seconds since 1/1/70
5 // The second 2 words are combined to form a ns offset to the above
6 static const int GPSTIMESTAMP_OFFSET = EVENT8FTIMESTAMP_OFFSET + 4;
7
8 int unscale(const double& d)
9 {
10 // double temp_needed_to_avoid_gcc_bug = d*32767./10.;
11 // return (int)temp_needed_to_avoid_gcc_bug;
12 return (int)(d*32767.001/10.);
13 }
14 // upper-lower to integer
15 int ul2int(const double& d1, const double& d2)
16 {
17 int id1 = unscale(d1);
18 int id2 = unscale(d2);
19 return (id1<<16)+(id2&0xffff);
20 }
21 int RawBeamSwicData::VmeSeconds() const
22 {
23 const double* data = fData.GetData();
24 return ul2int(data[GPSTIMESTAMP_OFFSET],
25 data[GPSTIMESTAMP_OFFSET+1]);
26 }
27 int RawBeamSwicData::VmeNanoseconds() const
28 {
29 const double* data = fData.GetData();
30 return ul2int(data[GPSTIMESTAMP_OFFSET+2],
31 data[GPSTIMESTAMP_OFFSET+3]);
32 }

```

5 Appendix: `ifbeamread.C`

```

1 // g++ wide_loop_devices.cc -o wide_loop_devices -I$IFBEAM_DIR/inc -L$IFBEAM_DIR/lib -lifbeam
2
3 #include <fstream>
4 #include <sstream>
5 #include <iostream>
6 #include <iomanip>
7 #include <stdarg.h>
8 #include <stdlib.h>
9 #include <stdio.h>
10 #include <string.h>
11 #include <errno.h>
12 #include "ifbeam.h"
13 #include "../util/utills.h"
14 #include "../util/WebAPI.h"
15 #include <math.h>
16 #include <map>
17

```

```

18 bool printMissing = false;
19
20 //
21 // this should throw an exception if no data exists for the time
22 //
23 void
24 checkdata(double t2) throw (WebAPIException) {
25     BeamFolder cf("NuMI_all","",10);
26     cf.FillCache(t2);
27 }
28
29 std::vector<std::string> UniqueNames(std::vector<std::string> names)
30 {
31     //
32     // Weed out duplicate names (also print out duplicates)
33     //
34     std::map<std::string,int> namecnt;
35     for ( size_t iname = 0; iname < names.size(); ++iname ) {
36         namecnt[names[iname]]++;
37     }
38     if ( names.size() != namecnt.size() ) {
39         std::cout << "bf.GetDeviceList() returned "
40                 << names.size() << " devices, "
41                 << namecnt.size() << " unique"
42                 << std::endl;
43     }
44     // clear old list
45     names.clear();
46     // build new list from sorted map
47     std::map<std::string,int>::const_iterator ncitr = namecnt.begin();
48     for ( ; ncitr != namecnt.end(); ++ncitr ) {
49         names.push_back(ncitr->first);
50         int cnt = ncitr->second;
51     }
52     return names;
53 }
54
55 typedef int (*processor)(BeamFolder&, std::vector<std::string>&, double when);
56 //
57 // call processdata(bf, time) for all the times between t1 and t2.
58 //
59 int
60 scanwindow(double t1, double t2, processor process_data) {
61     BeamFolder bf("NuMI_all");
62     double tend;
63     checkdata(t2);
64     std::vector<double> times;
65     std::vector<std::string> devices;
66
67     bf.set_epsilon(0.05);
68     tend = t1;
69     while (tend < t2) {
70         std::cout << std::setiosflags(std::ios::fixed) ;
71         std::cout << "tend is now: " << tend << std::endl;
72         bf.FillCache(tend);
73         times.clear();
74         times = bf.GetTimeList();
75         devices = UniqueNames( bf.GetDeviceList() );
76         for (size_t i = 0; i < times.size(); i++) {
77             if (times[i] < t2)
78                 (*process_data)(bf, devices, times[i]);
79         }
80         tend = bf.GetCacheEndTime() + .0001;
81     }
82 }

```

```

83
84 int
85 printhp121(BeamFolder &bf, std::vector<std::string>& devices, double t) {
86     size_t nval = 0;
87     size_t ngooddev = 0;
88     for (size_t idev=0; idev < devices.size(); ++idev ) {
89         std::string devname = devices[idev];
90         if ( devname.find("[") != std::string::npos ) {
91             // vector
92             std::vector<double> vvalues;
93             double devtime;
94             try {
95                 vvalues = bf.GetNamedVector(t,devname, &devtime);
96                 nval += vvalues.size();
97                 ngooddev++;
98             } catch (WebAPIException &we) {
99                 if (printMissing) std::cout << "got exception:" << we.what() << "\n";
100             }
101         } else {
102             // single value device
103             double value, devtime;
104             try {
105                 std::string devname_at = devname + "@";
106                 bf.GetNamedData(t,devname_at, &value, &devtime);
107                 nval += 1;
108                 ngooddev++;
109             } catch (WebAPIException &we) {
110                 if (printMissing) std::cout << "got exception:" << we.what() << "\n";
111             }
112         }
113     } // loop over devices
114     std::cout << std::setiosflags(std::ios::fixed) ;
115     std::cout << "time: " << t << " read " << nval << " values from "
116         << ngooddev << " of " << devices.size() << " devices" << std::endl;
117
118     return 1;
119 }
120
121 int
122 main() {
123
124     double window = 3600;
125     double t1 = 1371571200.0; // 2013-06-18 16:00:00
126     double t2 = t1 + window;
127
128     scanwindow(t1, t2, printhp121);
129
130     std::cout << "t1 " << t1 << " window size " << window << std::endl;
131 }

```

References