Bigbite library

Olivier Gayou
Massachusetts Institute of Technology

July 29, 2005

Contents

This is the status of the Bigbite library as it stands now. Many modifications need to be implemented in the future, even to resolve some of our current problems. This manual is compatible with the version in CVS as of the date on the title page.

For information on the Hall A analyzer (now called PODD), see `http://hallaweb.jlab.org/root/index.html`.

All detectors in the Bigbite library ultimately derive from the **THaAnalysisObject** class. This class have based functions, such as **THaAnalysisObject::DefineVariables()**, **THaAnalysisObject::ReadDatabase()**, **THaAnalysisObject::Decode()**, **THaAnalysisObject::CoarseProcess()** and **THaAnalysisObject::FineProcess()**. Each of these functions have to be implemented in the different detector classes.

Chapter 1

Bigbite

1.1 The Bigbite apparatus

The **THaBigBite** class is derived from **THaSpectrometer**. Therefore a momentum and a central angle must be assigned to it in the `db_run.dat` database. It is declared in the replay script, and detectors can be added to it there. For example:

```
// Add BigBite
THaApparatus* BB = new THaBigBite("BB","BigBite");
gHaApps->Add( BB );

// Add the detector
BB->AddDetector(new THaSRCSet("src","SRC Tracking set"));
```

For E01-015, the detector that is attached to BB is the SRC set of detectors (see Sec. 1.2).

1.2 The SRC detector set

The SRC set of detectors consists of the trigger plane and the auxiliary plane (see Secs. 1.3 and 1.4). In the **THaSRCSet::Setup()** function, these two subdetectors are defined, with respective (and imposed) names of “`tp`” and “`ap`”.

The main purpose of the **THaSRCSet** class is to implement tracking using information from the two detectors, separated by 1m. Two sets of tracks are constructed in the **THaSRCSet::ConstructTracks()** and **THaSRCSet::ConstructPTracks()**. The two sets are constructed identically, but the second one has more stringent constraints on ADC and TDC values used to define a hit in the planes, in order to select protons. This worked well on elastic data, but not so well on Carbon.

The tracking algorithm is very simple, and illustrated in Fig. 1.1: the first loop goes over the auxiliary plane, and the second loop over the E plane. For each combination of paddles, it looks whether there is a hit in the Aux paddle, a hit in the E paddle and a hit in either one of the overlapping δE paddles. If 3 hits are recorded, a track is defined. There is a maximum of 15 tracks.

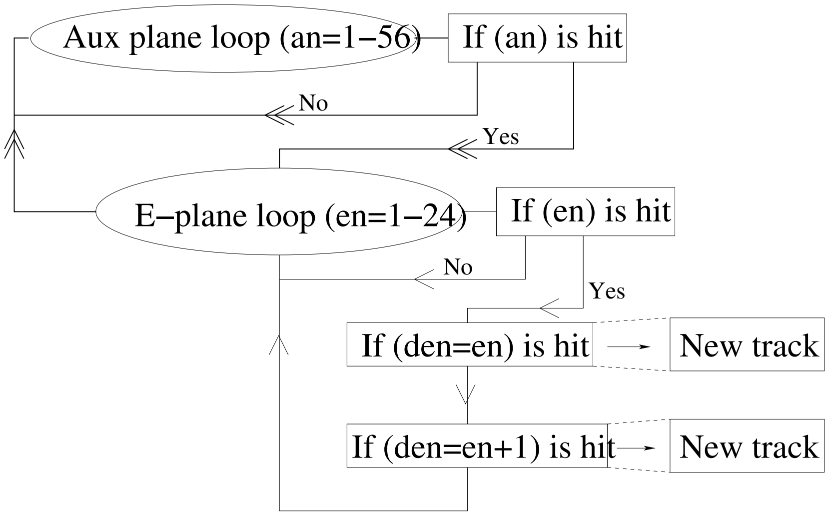

Figure 1.1: Tracking algorithm for Bigbite

When a track is created, the **THaSRCSet::MakeTrack()** function is called, creating all the array variables linked to the output. This is where all physical quantities are created (momentum, angle, etc...), using the paddle numbers or the time of flight, using algorithms written by Ran and that can be rather obscure to me.

There is a database associated to **THaSRCSet** objects, usually called `db_BB.src.dat`. It contains the values of the cuts on ADCs and TDCs for the second tracking (for proton in elastic case). It also contains matrix elements, that are currently set to 0 as the optics has not been fully developped yet.

1.3 The scintillator plane class

There is a **THaTriggerPlane** class, that simply defines an E and a δE plane, named respectively “`e`” and “`de`”. These planes are objects of class **THaScintillatorPlane**.

THaScintillatorPlane is also used for the neutron detector planes. For the ND planes, the TDCs were mutlihit, so the readout of multihit TDCs was implemented in this class, without affecting the readout of single hit TDC for Bigbite. The **PaddleMultiHits()** class at the beginning of the `THaScintillatorPlane.h` file defines paddle hits as combinations of left and right hits in the TDC in one paddle. For a single hit TDC detector, the number of hits is just limited to 1.

THaScintillatorPlane derives from **THaPlane**, which itself derives from **THaNonTrackingDetector**. The **THaPlane** class was created at a time where I thought it might be useful to have such a base class, however I think now that this step is not necessary, since it doesn't do anything more than **THaNonTrackingDetector**.

1.3.1 Database

There is of course a database associated to the plane. It is named, for example, for the E plane, `db_BB.src.tp.e.dat`. The data base contains, in order:

- the number of detectors in the plane;
- the detector map, where each line corresponds to a set of TDC channels; the format is crate, slot, first TDC in the set, last TDC and detector number of first channel; all ADCs come before TDCs, detector number 1 to N are for left paddle, and N+1 to 2N are for right paddle;
- the TDC offsets;
- the ADC pedestals;
- the ADC gains;
- the offset and scale parameters for the y-position calibration in terms of left−right
- the offset and scale parameters for the TOF calibration in terms of left+right;
- the x, y and z position of each detector, where $\hat{z}$ is along the central ray, $\hat{x}$ points down, perpendicular to $\hat{z}$ and $\hat{y}$ points to the left.

1.3.2 Decode

In the **THaScintillatorPlane::Decode()** function, the ADC and TDC arrays are read out, and some variables are filled. The first set of variables is the basic `fLT`, `fRT`, `fLA` and `fRA`, corresponding to hit number 0 (which is arbitrary in case of multihit TDCs).

The multihit arrays defined in the **PaddleMultiHits** class are also filled with the possible multihit TDCs. A paddle is said to be "hit" (`fIsHit[i]=1`) if there is at least one hit in the left and right TDC, and if the left and right ADC values corrected for pedestal are more than zero. Then, for each possible combination of left hits and right hits, the TOF is calculated for that hit, according to:

$$TOF = TOFscale * (L + R)/2 + TOFoffset$$

where $TOFscale$ and $TOFoffset$ are read from the database. Then the y-position is calculated, according to:

$$y = 100\frac{(L-R) + Yoffset - \frac{Yscale}{2}}{Yscale}$$

where $Yoffset$ and $Yscale$ are read from the database. y is expressed in cm. A test on y is then imposed ($|y| < 100$) to ensure that the hit is reasonably within the bar. If so, a new hit is defined as an object of class **THaScinHit**. Such an object has a TOF, y-position and energy associated to it, as well as the number of the paddle in question. Note the use of **THaScintHit** objects is more extensive in the Neutron Arm analysis, as it is useful for the multihit TDCs.

1.4 The auxiliary plane class

The **THaAuxPlane** class is very similar to the **THaScintillatorPlane** class, except that there is only one side of the bar that is connected to a PMT, and there is no multihit capability.

Chapter 2

Neutron arm

2.1 The Neutron Arm apparatus

2.1.1 Declaration

The **THaNA** apparatus class derives directly from the **THaApparatus** class. A distance from the target and angle must be assigned to it in the `db_run.dat` database. It is declared in the replay script the following way:

```
// Add Neutron Arm
THaApparatus* NA = new NA("NA","Neutron Arm");
gHaApps->Add( NA );
```

Detectors are added to it inside the class, because tracking depends on both the Neutron Detector and the Veto. They are added with the respective names “`nd`” and “`veto`”.

2.1.2 Tracking

In the **THaNA::Reconstruct()** method, the tracks from the Neutron Detector (see Sec. 2.2) are matched with Veto information to create charged and neutral tracks arrays. In a dual loop, over the ND tracks first, then over the Veto paddles, the Veto region around the ND track is checked for a hit (see Sec. 2.3 for Veto geometry). If the corresponding Veto region is hit, then the track is added to the “`wv`” track array (for “with veto”, i.e charged track), and if it is not, it is added to the “`wov`” track array (for “without veto”, i.e neutral track). This method still need to be improved, as it did not give really satisfactory results online with deuterium data. A track array identical to the ND track array, i.e ignoring the Veto information is also defined in this class.

2.2 The Neutron Detector class

The four planes of the ND are hardcoded in this class as **THaScintillatorPlane** objects, called "`p1`", "`p2`", "`p3`" and "`p4`". There is no database associated to the ND other than the individual plane databases, i.e there is no `db_NA.nd.dat` file.

The tracking algorithm for the ND is in the **THaNeutronDetector::FineProcess()** method. It is illustrated in Fig. 2.1. A first loop goes over the plane, from 1 to 4, and in each plane, a loop goes over the hits (which are read in the **THaScintillatorPlane::Decode()** function from top to bottom of the plane). Each hit is compared, in both angles, to all the previously defined tracks. If the hit has the same angles as a previously defined track, then it is added to the track. If not then a new track is created with this hit. A track is an object of class **THaNDTrack**.

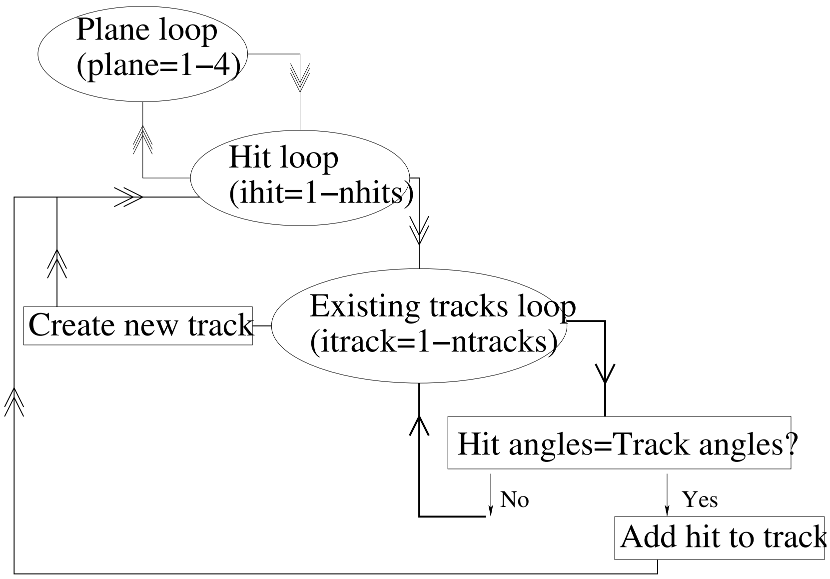

Figure 2.1: Tracking algorithm for Neutron detector (before Veto information is processed)

The maximum angle between the hit and the previous track to determine whether the hit belongs to the track is the quantity `fMinAngle[p]`. It is defined in **THaNeutronDetector::Init()**, as twice the angle between the two middle bars of the plane (therefore it is different for each plane, depending on the distance plane-target). The formula is:

$$MinAngle = 2\frac{x(middle) - x(middle-1)}{z(middle)}$$

Note that the same angle is used for the in-plane angle comparison. This should probably be changed, based on y-position resolution.

2.3 The Veto Class

The **THaVeto** class has to be different from the **THaScintillatorPlane** class, because the paddles are actually two "half"-paddles, overlapping in the middle. Each of these half-paddle is read by one PMT. Therefore, we can not compute the y-position from TDC, but we can define what side of the veto has (or has not) fired. The format of the `db_NA.veto.dat` database is the following:

- Number of paddles (1 paddle = 2 half-paddles!)
- Overlap region in the mdiddle (0.3 m for SRC)
- Detector map
- TDC offsets
- ADC pedestals
- ADC gains
- x, y and z positions of the paddles

At the time the database is readout, the angles of the edges of the overlap region. These angles are used in the **THaNA::Reconstruc()** method to check whether a ND track has a veto hit on its path.

The hits are decoded in the way similar to the **THaScintillatorPlane** objects, as multihit TDCs, but simpler, without the use of the **THaScintHit** class. The condition for a hit in a paddle is an OR of Left and Right, instead of an AND. Also, some variables are added to define when one side only is hit.

At present there is no TOF associated to the Veto detector.

Now that I write this, I realize that the algorithm to associate the ND and veto information is probably over-complicated. It might be useful to rewrite the **THaVeto** class where the members are the half-paddles instead of the paddles. I don't know.

Chapter 3

Physics modules

On top of the apparatus and detector classes, two physics modules were written, in a parallel way, one for Bigbite and the other for the Neutron Arm. Their goal was, at the time of writing during data taking, to quickly improve the selection of which track was the “golden” one in each apparatus. It is based on the angular correlation with the p_{miss} vector defined by the two HRSs in the **THaSecondaryKine**. Note that for this purpose, the line

```
#include ‘‘THaPrimaryKine.h’’
```

must be added in the `THaSecondaryKinematics.h` header file, which is not done in the release version of the Podd, nor in the CVS version of Podd. Please modify it in your local version to which you attach the Bigbite library.

The modules **THaTripleCoincidenceNeutron** and **THaTripleCoincidenceProton** are declared in the replay script in the following way:

```
   THaPhysicsModule* TC_n = new THaTripleCoincidenceNeutron(‘‘TC_n’’,
’’Neutron Triple Coincidence’’,’’SK’’,’’NA’’);
   gHaPhysics->Add(TC_n);

   THaPhysicsModule* TC_p = new THaTripleCoincidenceProton(‘‘TC_n’’,
’’Proton Triple Coincidence’’,’’SK’’,’’BB’’);
   gHaPhysics->Add(TC_p);
```

In **THaTripleCoincidenceNeutron::Process()**, the opposite of the initial momentum of the struck proton is computed, as:

$$p_i = -(x - q)$$

where x is the 4-momentum of the struck proton in the R-RHS and q is the 4-momentum transfer. Then the 3-momentum of this vector is extracted and rotated to the NA frame. In the loop over the Neutron Arm tracks, the NA track that is the closest to the $\vec{p}_i$ vector in angle is selected as the “golden” track. Cuts can then be made by the user on the angles of this track, and on the angle between p_i and this track.

The same applies to **THaTripleCoincidenceProton**.

Note that this class was written initially for the Neutron Arm analysis on Deuterium target, where p_i is well defined. A very selective cut on the angle between the track and p_i could be made ($\cos\theta > 0.98$) in order to clean the triple coincidence spectrum. This is not necessarily the case on heavier target, in particular due to Fermi momentum.

Appendix A

List of variables

A.1 SRC set

Name	Definition	Type	Index over
`BB.src.ntracks`	Number of tracks defined	scalar	
`BB.src.ne[i]`	E-bar number for a track	array	track number
`BB.src.nde[i]`	δE-bar number for a track	array	track number
`BB.src.na[i]`	Auxiliary bar number for a track	array	track number
`BB.src.xpos[i]`	*not too sure: Ran?*	array	track number
`BB.src.ypos[i]`	Transverse position in E-plane	array	track number
`BB.src.zpos[i]`	*undefined*		
`BB.src.theta[i]`	Out-of-plane angle of the track	array	track number
`BB.src.phi[i]`	In-plane angle of the track	array	track number
`BB.src.tof[i]`	TOF between aux and E planes	array	track number
`BB.src.etime[i]`	TOF to the E-plane (from the target)	array	track number
`BB.src.detime[i]`	TOF to the δE-plane	array	track number
`BB.src.atime[i]`	TOF to the Aux-plane	array	track number
`BB.src.time[i]`	Average of 3 previous TOFs	array	track number
`BB.src.dist[i]`	Distance between E- and Aux-bars	array	track number
`BB.src.tpath[i]`	Distance between E-bar and target	array	track number
`BB.src.ptof[i]`	Momentum calculated from TOF	array	track number
`BB.src.ptofT[i]`	*not too sure: Ran?*	array	track number
`BB.src.phit[i]`	Momentum from hit pattern	array	track number
`BB.src.pid[i]`	Test on ADC values in E and δE	array	track number
`BB.src.Ela[i]`	Corrected left ADC in E-bar	array	track number
`BB.src.Era[i]`	Corrected right ADC in E-bar	array	track number
`BB.src.aa[i]`	Corrected ADC in Aux-bar	array	track number
`BB.src.dEla[i]`	Corrected left ADC in δE-bar	array	track number
`BB.src.dEra[i]`	Corrected right ADC in δE-bar	array	track number
`BB.src.ttMom[i]`	Momentum from hit pattern (from file)	array	track number
`BB.src.ttTheta[i]`	OOP-angle from hit pattern	array	track number
`BB.src.nptracks[i]`	see above, with extra tests on ADCs	array	track number

There are also the same set of variables (with a 'p') at the end, for the second set of tracks defined with more stringent tests on ADC and TDC values (see Sec. 1.2).

A.2 Scintillator plane

General variables:

Name	Definition	Type	Index over
BB.src.tp.e.nlthit	Number of left paddles with TDC hit	scalar	
BB.src.tp.e.nrthit	Number of right paddles with TDC hit	scalar	
BB.src.tp.e.nlahit	Number of left paddles with ADC hit	scalar	
BB.src.tp.e.nrahit	Number of right paddles with ADC hit	scalar	
BB.src.tp.e.lt[i]	Left raw TDC value in a paddle	array	paddle number
BB.src.tp.e.rt[i]	Right raw TDC value in a paddle	array	paddle number
BB.src.tp.e.la[i]	Left raw ADC value in a paddle	array	paddle number
BB.src.tp.e.ra[i]	Right raw ADC value in a paddle	array	paddle number
BB.src.tp.e.lt_c[i]	Left corrected TDC value in a paddle	array	paddle number
BB.src.tp.e.rt_c[i]	Right corrected TDC value in a paddle	array	paddle number
BB.src.tp.e.la_c[i]	Left corrected ADC value in a paddle	array	paddle number
BB.src.tp.e.ra_c[i]	Right corrected ADC value in a paddle	array	paddle number
BB.src.tp.e.tdiff[i]	Left − Right TDC in a paddle	array	paddle number
BB.src.tp.e.ypos[i]	y-position in paddle from tdiff	array	paddle number
BB.src.tp.e.ybar[i]	y-position of the paddle	array	paddle number
BB.src.tp.e.xbar[i]	x-position of the paddle	array	paddle number
BB.src.tp.e.zbar[i]	z-position of the paddle	array	paddle number
BB.src.tp.e.tof[i]	TOF to paddle from (L+R)	array	paddle number
BB.src.tp.e.npadhit	Number of paddles hit	scalar	
BB.src.tp.ishit[i]	= 1 if paddle is hit, 0 otherwise	array	paddle number

Of course the same applies to `BB.src.tp.de`, as well as `NA.nd.p1`, ...

Multihit variables:

Name	Definition	Type	Index over
NA.nd.p1.nhit	Total number of hits in the plane	scalar	
NA.nd.p1.mypos[i]	y-position of hit	array	hit number in plane
NA.nd.p1.mtof[i]	TOF of hit	array	hit number in plane
NA.nd.p1.pad[i]	Paddle number of hit	array	hit number in plane
NA.nd.p1.nhitspadl	Number of hits in left paddle	scalar	
NA.nd.p1.nhitspadr	Number of hits in right paddle	scalar	

A.3 Auxiliary plane

Name	Definition	Type	Index over
`BB.src.ap.nthit`	Number of paddles with TDC hit	scalar	
`BB.src.ap.nahit`	Number of paddles with ADC hit	scalar	
`BB.src.ap.t[i]`	Raw TDC value in a paddle	array	paddle number
`BB.src.ap.a[i]`	Raw ADC value in a paddle	array	paddle number
`BB.src.ap.t_c[i]`	Corrected TDC value in a paddle	array	paddle number
`BB.src.ap.a_c[i]`	Corrected ADC value in a paddle	array	paddle number

A.4 Neutron Arm

Name	Definition	Type	Index over
`NA.ntracks`	Total number of tracks	scalar	
`NA.tr.theta[i]`	Out-of-plane angle of track	array	track number
`NA.tr.phi[i]`	In-plane angle of track	array	track number
`NA.tr.nhits[i]`	Number of hits defining the track	array	track number
`NA.tr.tof[i]`	TOF from target to first plane hit	array	track number
`NA.tr.energy[i]`	Sum of all ADCs associated with the track	array	track number
`NA.tr.firstpad[i]`	Paddle number of the first hit of the track	array	track number
`NA.tr.firstplane[i]`	Plane number of the first hit of the track	array	track number
`NA.wv.ntracks`	Total number of tracks	scalar	
`NA.tr.wv.theta[i]`	Out-of-plane angle of charged track	array	charged track number
`NA.tr.wv.phi[i]`	In-plane angle of charged track	array	charged track number
`NA.tr.wv.nhits[i]`	Number of hits defining the charged track	array	charged track number
`NA.tr.wv.tof[i]`	TOF from target to first plane hit	array	charged track number
`NA.tr.wv.energy[i]`	Sum of all ADCs associated with the charged track	array	charged track number
`NA.tr.wv.rveto[i]`	= 1 if the right half of the veto is hit, 0 otherwise	array	charged track number
`NA.tr.wv.lveto[i]`	= 1 if the left half of the veto is hit, 0 otherwise	array	charged track number
`NA.wov.ntracks`	Total number of tracks	scalar	
`NA.tr.wov.theta[i]`	Out-of-plane angle of neutral track	array	neutral track number
`NA.tr.wov.phi[i]`	In-plane angle of neutral track	array	neutral track number
`NA.tr.wov.nhits[i]`	Number of hits defining the neutral track	array	neutral track number
`NA.tr.wov.tof[i]`	TOF from target to first plane hit	array	neutral track number
`NA.tr.wov.energy[i]`	Sum of all ADCs associated with the neutral track	array	neutral track number

A.5 Neutron Detector

Name	Definition	Type	Index over
`NA.nd.nplaneshit`	Number of planes hit	scalar	
`NA.nd.ntracks`	Total number of tracks	scalar	
`NA.nd.tr.theta[i]`	Out-of-plane angle of track	array	track number
`NA.nd.tr.phi[i]`	In-plane angle of track	array	track number
`NA.nd.tr.nhits[i]`	Number of hits defining the track	array	track number
`NA.nd.tr.tof[i]`	TOF from target to first plane hit	array	track number
`NA.nd.tr.energy[i]`	Sum of all ADCs associated with the track	array	track number

Note that the `NA.nd.tr` variables and the `NA.tr` variables are redundant.

A.6 Veto

Name	Definition	Type	Index over
`NA.veto.nhit`	Total number of hits in the veto plane	scalar	
`NA.veto.pad`	Paddle number of hit	array	hit number
`NA.veto.nlthit`	Number of hits (from TDC) in the left paddles	scalar	
`NA.veto.nrthit`	Number of hits (from TDC) in the right paddles	scalar	
`NA.veto.nlahit`	Number of hits (from ADC) in the left paddles	scalar	
`NA.veto.nrahit`	Number of hits (from ADC) in the right paddles	scalar	
`NA.veto.lt`	Raw left TDC signal (only hit 0 in multihit TDCs)	array	paddle number
`NA.veto.rt`	Raw right TDC signal (only hit 0 in multihit TDCs)	array	paddle number
`NA.veto.la`	Raw left ADC signal	array	paddle number
`NA.veto.ra`	Raw right ADC signal	array	paddle number
`NA.veto.lt_c`	Corrected left TDC signal (only hit 0 in multihit TDCs)	array	paddle number
`NA.veto.rt_c`	Corrected right TDC signal (only hit 0 in multihit TDCs)	array	paddle number
`NA.veto.la_c`	Corrected left ADC signal	array	paddle number
`NA.veto.ra_c`	Corrected right ADC signal	array	paddle number
`NA.veto.ybar`	y-position of paddle	array	paddle number
`NA.veto.xbar`	x-position of paddle	array	paddle number
`NA.veto.zbar`	z-position of paddle	array	paddle number
`NA.veto.npadhit`	Number of paddles hit	scalar	
`NA.veto.ishitl`	= 1 is the left paddle is hit, 0 otherwise	array	paddle number
`NA.veto.ishitr`	= 1 is the right paddle is hit, 0 otherwise	array	paddle number
`NA.veto.hitregion`	*unused*	scalar	

A.7 Triple Coincidence Neutron

Name	Definition	Type	Index over
`TC_n.pn_x_l`	x-comp of $\vec{p}_i$ in the lab	scalar	
`TC_n.pn_y_l`	y-comp of $\vec{p}_i$ in the lab	scalar	
`TC_n.pn_z_l`	z-comp of $\vec{p}_i$ in the lab	scalar	
`TC_n.pn_x`	x-comp of $\vec{p}_i$ in the ND frame	scalar	
`TC_n.pn_y`	y-comp of $\vec{p}_i$ in the ND frame	scalar	
`TC_n.pn_z`	z-comp of $\vec{p}_i$ in the ND frame	scalar	
`TC_n.pn`	Magnitude of $\vec{p}_i$	scalar	
`TC_n.angle`	Cosine of the angle between $\vec{p}_i$ and the golden track	scalar	
`TC_n.ntof`	TOF of the golden track	scalar	
`TC_n.nphi`	In-plane angle of the golden track	scalar	
`TC_n.ntheta`	Out-of-plane angle of the golden track	scalar	
`TC_n.nenergy`	Energy deposited by the golden track	scalar	
`TC_n.nnhits`	Number of hits in the golden track	scalar	

A.8 Triple Coincidence Proton

Name	Definition	Type	Index over
TC_p.pp_x_l	x-comp of $\vec{p}_i$ in the lab	scalar	
TC_p.pp_y_l	y-comp of $\vec{p}_i$ in the lab	scalar	
TC_p.pp_z_l	z-comp of $\vec{p}_i$ in the lab	scalar	
TC_p.pp_x	x-comp of $\vec{p}_i$ in the ND frame	scalar	
TC_p.pp_y	y-comp of $\vec{p}_i$ in the ND frame	scalar	
TC_p.pp_z	z-comp of $\vec{p}_i$ in the ND frame	scalar	
TC_p.pp	Magnitude of $\vec{p}_i$	scalar	
TC_p.angle	Cosine of the angle between $\vec{p}_i$ and the golden track	scalar	
TC_p.ptof	TOF of the golden track	scalar	
TC_p.pphi	In-plane angle of the golden track	scalar	
TC_p.ptheta	Out-of-plane angle of the golden track	scalar	