TreeSearch Track Reconstruction for BigBite Software User Guide v1.0

Jens-Ole Hansen
Jefferson Lab

March 12, 2010

1 Installation

1.1 Requirements

The TreeSearch library is implemented as a plugin module for the Hall A `C++` analyzer Podd. Minimum software requirements are as follows:

- Linux platform. RedHat Enterprise Linux 3 or Fedora 6 or newer have been tested. Other Unix platforms will probaby work as well, but require modifications to the Makefile
- GNU make 3.7
- ROOT 5
- Podd 1.5

It is especially important to use at least Podd version 1.5. Older versions of Podd do not offer the database and detector map functionality required by the TreeSearch library.

1.2 Download

The source code can ba obtained via the Web from

```
http://hallaweb.jlab.org/root/TreeSearch/
```

or via CVS, using the commands

```
setenv CVS_RSH ssh
cvs -d :ext:cvs.jlab.org:/group/halla/analysis/cvs co TreeSearch
```

Web access is possible either directly from any JLab computer or by using the account name and password of the Hall A counting house analysis account. CVS access is permitted for anyone using a valid JLab CUE username and password.

1.3 Compilation

Compilation of the source code is straightforward:

- Ensure that `ROOT` and `Podd` are set up. The environment variables `$ROOTSYS` and `$ANALYZER` must be set.
- Change to the TreeSearch source directory.
- Edit the `Makefile` if desired. At the top of the `Makefile`, several variables can be set that modify the compilation:
  - `VERBOSE`: enable verbose status messages that can be enabled by setting the verbosity level with a call to `SetDebug()` at run time (recommended)
  - `TESTCODE`: enable additional computations and global variables containing detailed statistics information about the progress of the analysis. This option will cause a performance hit and export a large amount of internal information as global variables, which may greatly increase the output file size. Only recommended for experts.
  - `DEBUG`: compile a debug version for `gdb`. This will *significantly* slow down the analysis. Only recommended for troubleshooting by experts.
- Run `make` to compile the code. A shared library named `libTreeSearch.so` will be built. This library can be loaded into Podd using `gSystem->Load()`.

2 Configuration

To use the TreeSearch track reconstruction for replay, several items will need to be configured:

1. Crate map
2. Replay script
3. MWDC database file
4. Output definitions

These are discussed in the following.

2.1 Crate Map

The crate map file used for replay (usually `db_cratemap.dat`) must contain proper entries for the MWDC TDCs. The software supports both the old Fastbus readout (LeCroy 1877 TDCs) and the VME CAEN pipeline TDCs. The crate map is normally set up by the DAQ expert. There are no special considerations about the setup of the modules for the MWDC in the crate map.

With the present version of the TreeSearch library, some information from the regular DAQ crate map, `db_cratemap.dat`, must be duplicated in the database file. See the detailed description of the `cratemap` database key below. This requirement will be dropped in a future version.

2.2 Replay Script

The TreeSearch code is implemented as a tracking detector class, `TreeSearch::MWDC`, describing the BigBite MWDCs. For replay, the MWDC detector should be made part of a spectrometer apparatus, describing BigBite. For convenience, the library includes such an apparatus, called `TreeSearch::BigBite`, which contains only the MWDC detector named "mwdc". One can either use this apparatus and add additional detectors (*e.g.* scintillators, calorimeter) as needed, or use a different spectrometer class (implemented outside of the TreeSearch library) and add the `TreeSearch::MWDC` detector to it (using the `AddDetector()` method — see the Podd documentation for details).

As an example, we consider the former approach. Below are example commands that could be used in a replay script:

```
gSystem->Load("libTreeSearch.so");
THaSpectrometer* B = new TreeSearch::BigBite("B", "BigBite");
gHaApps->Add(B);
// optionally, add detectors
THaDetector* BS1 = new THaScintillator("BS1", "BigBite S1");
B->AddDetector(BS1);
```

This assumes that `libTreeSearch.so` is in your `$LD_LIBRARY_PATH`, for example via a symbolic link from the `$ANALYZER` directory. Otherwise, the full path of the library needs to be specified in the `gSystem->Load()` command.

2.3 MWDC Database File

The database file for the MWDC detector contains all configuration and calibration data. It must be carefully set up before replay. An example database file, named `db_B.mwdc.dat`, is included in the source code distribution. The file name follows the usual Podd conventions: `db_`*prefix*`.dat`, where *prefix* is the spectrometer name ("B") followed by a dot and the MWDC detector name ("mwdc").

Unlike many older Podd database files, the MWDC database may be freely formatted. Blank lines and anything between a comment character, `#`, and the end of a line is ignored. Lines may be continued with a trailing backslash character, \, for better readability. Valid lines contain key/value pairs of the form

```
key = value
```

In the present version of the software, all keys must begin with the prefix of the detector, *i.e.* the same string that distinguishes the database file name. In the above example, the prefix is "B.mwdc". Therefore, a database key "timecut" would be specified in the file as

```
B.mwdc.timecut = 1
```

The available database keys (without prefix) are described in detail in the following sections. Table 1 lists the properties of the individual keys. The table columns have the following meanings:

- "Key" is the name of the key.
- "Class" indicates the part of the program to which the key applies. "MWDC" keys apply to the entire MWDC detector class, *i.e.* represent global parameters; Projection ("Proj") keys refer to coordinate projections, *i.e.* u, v, x or y; and WirePlane ("WP") keys apply to individual wire planes, *e.g.* $u1$, $x2$, etc.

  Key names of class "Projection" or "WirePlane" must normally be prefixed first by the common detector prefix described above ("B.mwdc"), followed by the projection or wire plane name to which they apply, for example "B.mwdc.u1.detmap". However, see the description of the "Up-level" column below.
- "Up-level" indicates whether a global value for a per-projection or per-wire plane key may be set. To set such a value, one omits the projection- or plane-specific part of the key's name when specifying the key in the database. A value set in this way applies to all projections or wire planes, respectively, except for projections of planes for which values are given explicitly. For instance, if all wire planes had 200 wires, except for plane $x1$, which had 240, the most efficient way to specify this configuration in the database would be

  ```
  B.mwdc.nwires     =  200
  B.mwdc.x1.nwires  =  240
  ```

  Global defaults are not available where they clearly do not make sense, for example the detector map of a plane, which must be different for every plane.
- "Type" indicates the data type (integer, string, floating point, array, matrix). A type of "array" means a one-dimensional vector of data; a type of "matrix" represents a rectangular two-dimensional matrix of data, where the number of matrix columns and their respective meanings is explained in the detailed description of the key.
- "Required" indicates whether the respective key is required to be present. If it is not required and omitted, a default value is used, which is given in the next column.
- "Default" (optional keys only) gives the value that is used if the key is omitted from the database. A default of "auto" indicates that a suitable value is automatically calculated based on other information. See the respective key's description for details.

Key	Class	Up-level	Type	Required	Default
planeconfig	MWDC	N	String	Y	–
cratemap	MWDC	N	Integer Matrix	Y	–
calibrate	MWDC	N	String	N	empty
3d_maxmiss	MWDC	N	Integer	N	auto
3d_matchcut	MWDC	N	Float	N	10^{-4}
3d_chi2_conflevel	MWDC	N	Float	N	10^{-9}
maxthreads	MWDC	N	Integer	N	auto
timecut	MWDC	N	Boolean	N	T
pairsonly	MWDC	N	Boolean	N	F
nopartner	MWDC	N	Boolean	N	F
event_display	MWDC	N	Boolean	N	F
disable_tracking	MWDC	N	Boolean	N	F
disable_finetrack	MWDC	N	Boolean	N	F
search_depth	Proj	Y	Integer	Y	–
angle	Proj	N	Float	N	auto
maxslope	Proj	Y	Float	N	auto
cluster_maxdist	Proj	Y	Integer	N	0
min_fit_planes	Proj	Y	Integer	N	3
chi2_conflevel	Proj	Y	Float	N	10^{-3}
maxmiss	Proj	Y	Integer	N	0
req1of2	Proj	Y	Boolean	N	T
maxpat	Proj	Y	Integer	N	no limit
detmap	WP	N	Integer Matrix	Y	–
nwires	WP	N	Integer	Y	–
wire.pos	WP	N	Float	Y	–
wire.spacing	WP	Y	Float	Y	–
xp.res	WP	Y	Float	Y	–
ttd.converter	WP	Y	String	Y	–
ttd.param	WP	Y	Float Array	Y	–
tdc.offsets	WP	N	Float Array	Y	–
description	WP	N	String	N	auto
type	WP	N	String	N	auto
required	WP	N	Boolean	N	F
drift.min	WP	Y	Float	N	no limit
drift.max	WP	Y	Float	N	no limit
maxhits	WP	Y	Integer	N	no limit

Table 1: MWDC database key properties. See text for description of columns.

2.3.1 planeconfig

This is a string of the space-separated names of all the wire planes that should be included in the analysis. For each name, a WirePlane object of the same name will be created. Normally, the minimum number of planes required is 9 — three planes in each of three projections. Plane-specific database entries need to be created for each plane named in this configuration string.

Names can be arbitrary strings, although it is advantageous to use names corresponding to the actual configuration of the detector, such as *x1*, *u2*, etc.

By default, the first character of each plane's name indicates that plane's projection type. For instance, a plane named *u1* will be automatically associated with the u-projection. Its wires will be assumed to be oriented according to the angle of the u-coordinate (see `angle` key). This behavior can be overridden using the `type` key. The projection type must be one of the supported types (u, v, x, or y).

Furthermore, it is assumed that wire planes are arranged in pairs (partners) of adjacent wire chambers whose wires are offset by one-half wire spacing. To indicate that one plane is to be considered the partner of another, the partner plane's name must be identical to that of the other plane except for a trailing p (for "prime"). Example: *x1* and *x1p* will be considered partner planes. Plane partnering can currently only be configured via the appended p character in the name. It can be turned off via the `nopartner` key.

2.3.2 nopartner

If set, completely turns off partnering of wire planes. Partnering is enabled by default.