New Track Event Model HowTo

Jose A. Hernando, E. Rodrigues

LHCb Tracking Workshop, Orsay, 19th May 2005

* List of packages available

* How to get started
  - practicalities
  - finding information
  - requirements files

* Guidelines

* HowTo’s
List of Packages

Available in CVS

- Event/
  - TrackEvent
- Trg/
  - TrackExtrapolators
  - TrgConverters
  - LHCbKernel
  - TrackProjectors
  - LHCbInterfaces
- TrackTools
- Tr/
  - TrackFitEvent
  - TrackFitter
  - TrackIdealPR
  - TrConverters
  - TrackPython

(about 70 files adapted / invented … more to come …)

-> be aware that some repackaging is to be made (c.f. yesterday)
How to get started?

Practicalities

- Packages of new event model not yet part of official LHCb software releases
  - Exceptions: Kernel/Xxx, Event/TrackEvent
  - To be done (very) soon
- Need to getpack each package required in user code
- Code is evolving / being modified, improved, etc.
  - expect need for regular “getpack’s”

Finding information

- Doxygen documentation of “at-present” classes and algorithms
  Regularly updated at
  http://cern.ch/eduardo.rodrigues/lhcb/tracking/event_model
- CVS repository is the place to check for latest versions
- Jose and myself are always happy to answer questions/doubts/…
Requirements files

// if access needed to Tracks
use TrackEvent v* Event

// if access needed to XxxMeasurements
use TrackFitEvent v* Tr

// for using general tools, extrapolators, …
use TrackExtrapolators v* Tr
use TrackTools v* Tr
Guidelines

Tracks

• Base class for tracks

• Other track classes may inherit from it, say internally in pattern recognition algorithms, if really needed
  • Should be avoided as much as possible …
  • Additional features may be introduced in the base class, instead?

• Main source of information (see later)
  • No need to go through the states as in old event model
  • “physics state” for getting $p$, $p_t$, …, in many practical cases

States

• Internal representation of the track, at different positions

• Not need in most cases
  • The extrapolators do the job for you (see later)
**Guidelines**

**Measurements**

- Used mainly in fitting code
- Internal, not stored on DST
- Can be (re)produced from the LHCbIDs, stored on the DST
- Dedicated measurements for each sub-detector (e.g. OTMeasurement) are dealt with by the dedicated projectors (see later)
  - User is encouraged to use the base class Measurement, together with the TrackMasterProjector
Nodes

- Only stored in the tracks during fitting
- Not stored on DST
- Store the connection / relation between a State and a Measurement
Guidelines

Extrapolators

- A variety of extrapolators, adapted and extended from the old model
- Useful for getting track info at a certain position (z, plane)
- User passes a track as an argument; it gets a state
  - Makes available: position, momentum, covariance matrix, etc.
- TrackMasterExtrapolator delegates the work
Guidelines

Projectors

• Project a state onto a measurement
• TrackOTProjector, VeloRProjector, etc. for dedicated XxxMeasurements
• User does not need to care about the details
  • TrackMasterProjector figures out which TrackXxxProjector it needs for you
• Place where “local* (Measurements) and “global” (States) information
  is brought together
  • Main part of tracking software where the technical problems related to
    (mis)alignment are dealt with
Guidelines

Ideal pattern recognition TrackIdealPR

- Ideal pattern recognition adapted to work with new model
- Main algorithm for testing projectors, extrapolators, fitting, …
  - First users got already their hands dirty with it: Jacopo, Edwin, …?
HowTo’s
Tracks* tracksCont = get<Tracks>("/Event/Rec/Track/Ideal");

dump() << "Tracks container contains " << tracksCont -> size() << " tracks" << endreq;

Tracks::const_iterator iTrk;
for ( iTrk = tracksCont->begin(); tracksCont->end() != iTrk; ++iTrk ) {
  dump() << "-> Track # " << (*iTrk) -> key() << endreq
  << "  charge                  = " << (*iTrk) -> charge() << endreq
  << "  is Valid                 = " << (*iTrk) -> checkFlag( TrackKeys::Valid ) << endreq
  << "  is Unique             = " << (*iTrk) -> checkFlag( TrackKeys::Unique ) << endreq
  << "  is of type              = " << (*iTrk) -> type() << endreq
  << "  is Backward         = " << (*iTrk) -> checkFlag( TrackKeys::Backward ) << endreq
  << "  # measurements = " << (*iTrk) -> nMeasurements() << endreq;
// ...
}
// from TrackEvent
#include "Event/TrackKeys.h"
#include "Event/StateKeys.h"

... Tracks::const_iterator iTrk;
for ( iTrk = tracksCont->begin(); tracksCont->end() != iTrk; ++iTrk ) {
    debug()
    "-> Track # " "("iTrk) -> key() " endreq
    " * from algorithm = " "("iTrk) -> history() " endreq
    " * Kalman fitted? = " "("iTrk) -> checkHistoryFit( TrackKeys::Kalman ) " endreq
    " * has State at location BegRich1? = " "("iTrk) -> hasStateAt( StateKeys::BegRich1 ) " endreq;

    HepPoint3D pos;
    HepVector3D mom;
    HepSymMatrix cov6D;
    // position and momentum of the “physics state” (i.e. the one stored on the DST)
    StatusCode sc = (*iTrk) -> positionAndMomentum( pos, mom, cov );
    ...
}
Extrapolating a track

```cpp
// from TrackTools
#include "TrackTools/ITrackExtrapolator.h"

... 
ITrackExtrapolator* m_extrapolator;

//Retrieve TrackExtrapolator tool
m_extrapolator = tool<ITrackExtrapolator>( "TrackHerabExtrapolator" );

... 
Tracks::const_iterator iTrk;
for ( iTrk = tracksCont->begin(); tracksCont->end() != iTrk; ++iTrk ) {
    ...
    double z = 3000.;
    State myState;
    HepPoint3D plane;
    //position and momentum of the "physics state" (i.e. the one stored on the DST)
    StatusCode sc = m_extrapolator -> propagate( **iTrk, z, myState );
    if ( sc.isSuccess() ) {
        debug() << " - state at z = " << z << " has slopes " << myState.slopes() << endreq;
    }
    ...
} 
```