**New Track Event Model HowTo**

Jose A. Hernando, E. Rodrigues

LHCb Software Week, CERN, 23rd May 2005

* How to get started
  - practicalities
  - finding information

* Some guidelines

• HowTo’s
Practicalities

- Packages of new event model not yet part of official LHCb software releases
  - Exceptions: LHCbID.h in Kernel/LHCbKernel, Event/TrackEvent
- Working versions of all packages (done so far) for end of week
- Then all packages to go into next software release (thanks Marco)
- Plan to follow the official releases with updates, etc. ...
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 where to check for latest versions

• Twiki pages of Track Event Model Task Force at
  https://uimon.cern.ch/twiki/bin/view/LHCb/LHCbTrackModelTaskForce

• Jose and myself are always happy to answer questions/doubts/…
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 – in most cases - to go through the states as in old event model
  - “first state” (the one always stored on DST) for getting p, pt, …, in many practical cases

States

- Internal representation of the track, at different positions
- Not need in most cases
  - The extrapolators do a lot of the job for you (see later)
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
  - Is still called TrackFirstCleverExtrapolator – to be changed
Ideal pattern recognition package: Tr/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
  - You can be the next …
Side remarks

• We made the choice of passing references as arguments to methods
  • No need to take care about deletion of objects
  • E.g.: natural thing to do in tools (such as extrapolators) that do some job with a track but do not get ownership, etc.

• “clone” methods return pointers
  • Since the user is then naturally responsible for what it clones
  • User is responsible for deleting the objects cloned
// .cpp file

Tracks* tracksCont = get<Tracks>("/Event/Rec/Track/Ideal");

debug() << "Tracks container contains " << tracksCont -> size() 
<< " tracks" << endl;

Tracks::const_iterator iTrk;
for ( iTrk = tracksCont->begin(); tracksCont->end() != iTrk; ++iTrk ) {
  Track& track = *(*iTrk);
  debug()
    << "-> Track # " << track.key() << endl
    << " * charge = " << track.charge() << endl
    << " * is of type = " << track.type() << endl
    << " * is Backward = " << track.checkFlag( TrackKeys::Backward ) << endl
    << " * # measurements = " << track.nMeasurements() << endl;
  // ...

  // position and momentum of the “first state” (i.e. the one stored on the DST)
  HepPoint3D pos;
  HepVector3D mom;
  HepSymMatrix cov6D;
  track.positionAndMomentum( pos, mom, cov );
  // ...
}

// .h file

// from TrackEvent
#include "Event/Track.h"
#include "Event/TrackKeys.h"
Tracks flags, history, ...

// .cpp file

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

// get the state closest to, say, z = 2000.
double z = 2000.;
State& aState = track.closestState( z );
...
}

// .h file

// from TrackEvent
#include "Event/TrackKeys.h"
#include "Event/StateKeys.h"
// .cpp file

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

...

Tracks::const_iterator iTrk;
for ( iTrk = tracksCont->begin(); tracksCont->end() != iTrk; ++iTrk ) {
    ...
    Track& track = *(*iTrk);
    double z = 3000.;
    State myState;
    // propagate the track to a z-position (getting all info via a State)
    StatusCode sc = m_extrapolator -> propagate( track, z, myState );
    if ( sc.isSuccess() ) {
        debug() << " state at position = " << myState.position() << endreq
            << " momentum = " << myState.momentum() << endreq
            << " transverse momentum Pt = " << myState.pt() << endreq;
        HepSymMatrix& cov6D = myState.posMomCovariance();
    }
}

// .h file

// from TrackInterfaces
#include "TrackInterfaces/ITrackExtrapolator.h"

...

ITrackExtrapolator* m_extrapolator;
// .cpp file

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

...
Tracks::const_iterator iTrk;
for ( iTrk = tracksCont->begin(); tracksCont->end() != iTrk; ++iTrk ) {
    ...
    Track& track = *(iTrk);
    double z = 3000.;
    // propagate the track to a z-position (directly getting all info without passing via the State - PREFERRED)
    HepPoint3D pos;
    HepVector3D mom;
    HepSymMatrix cov6D;
    StatusCode sc = m_extrapolator -> positionAndMomentum( track, z, pos, mom, cov6D );
    if ( sc.isSuccess() ) {
        debug() << " - track at z-position = " << z << endreq
        << " has 3D-position = " << pos << endreq
        << " momentum = " << mom << endreq;
    }
    ...
}

// .h file

// from TrackInterfaces
#include "TrackInterfaces/ITrackExtrapolator.h"
...
ITrackExtrapolator* m_extrapolator;