Current Track Fit TDS Output Structure

All declared explicitly in the TDS

Repeated 4 Times
Once per Type:
meas, pred, fit, smooth

Event::TkrKalFitTrack

Event::TkrFitPlaneCol
(ObjectVector<TkrFitPlane>)

ContainedObject

Event::TkrFitTrackBase

Event::TkrFitPlane

Event::TkrFitHit

Event::TkrFitPar

Event::TkrFitMatrix

Event::TkrFitMatrix
TkrKalFitTrack Considerations

- Change inheritance to:

  Class TkrKalFitTrack : public TkrFitTrackBase, TkrFitPlaneCol, ContainedObject

  - Current implementation causes problems for creating Gaudi converters.
  - Originally thinking was that one might want to create a new track fit, e.g. related to the neural net pattern recognition or something, and want a different output class with more/different output information. The idea of the current inheritance scheme was to make it more “plug and play.”

- Question for output track classes:
  - TkrFitPlaneCol is an ObjectVector containing information for each plane with measured hit information. Should this be augmented to include planes the track passes through but which do not have measured hits?
    - A bit tricky in how you define “plane”
TkrFitPlane Considerations

- Currently, TkrFitPlane has three purposes:
  - Store information particular to a given plane
    - Index to the cluster hit
    - Tower, Plane and projection information
    - Z coordinate of the plane
    - Energy of the track at this plane
    - Radiation Lengths
    - Active Distance
  - Store the TkrFitHits which contain track parameters and covariance matrices for each fit type
    - Measured, Projected, Filtered and Smoothed
    - Also Store the MS Covariance matrix at this plane as well
  - Provide methods to
    - Calculate energy of the track at this plane taking into account energy loss from previous plane
    - Calculate the contribution to the track $\chi^2$ at this plane
TkrFitHit Considerations

• Currently TkrFitHit has three data members
  – It’s type (measured, projected, filtered, smoothed)
  – A TkrFitPar representing the track parameters of the above type
  – A corresponding TkrFitMatrix containing the covariance matrix for the above

• Move VolumelIdentifer and z-coordinate to TkrFitHit object?
TkrFitPar & TkrFitMatrix Considerations

• Parameter storage or mathematical objects?
  – Convenient to define matrix multiplication on objects so copying to HepVector and HepMatrix is not required.
  – Restricts generalization of track fit code, must define operations to use TkrFitPar and TkrFitMatrix objects only.
    • In particular, current implementation restricts code to use 4 vectors and 4x4 matrices only
  – HepVector and HepMatrix more general but pay price in performance
    • HepVector and HepMatrix have minimum sizes pushed on stack when created
    • Indexed math operations not as fast as inline math defined on specific classes
      – Is this a driving issue?
  – Parameter Storage:
    • Simplify definitions of TkrFitPar and TkrFitMatrix
      – Instantiation, Set and Get methods only
      – Some savings in storage possible (e.g. TkrFitMatrix is always symmetric)
    • Must define copy operations to create and retrieve from objects upon which math operations are defined
    • Define a helper class which defines operations on TkrFitPar and TkrFitMatrix?
  – Mathematical Objects:
    • Need to augment math methods to include more operations
      – e.g. Vector multiplication with non-square Kalman projection matrices

• Other Considerations?
  – Do these need to be Gaudi DataObjects?
Random Thoughts

• We should convert to storing the VolumelIdentifier instead of Tower, Plane and projection separately
  – Will need to figure out how to make them human readable in the debugger
• The cluster index should be replaced with a “SmartRef” to point to the particular TkrCluster
  – Avoid having to index in TkrClusterCol
• How will TkrFitPar and TkrFitMatrix be accessed?
  – Would like to remove the projection processing that goes on in TkrRecon with calls to retrieve measured and non-measured coordinates. As they exist now, TkrFitPar and TkrFitMatrix do not have the information required to decipher this
  – A possible solution? These two objects are only accessible through TkrFitHit which will store the VolumelIdentifier?
• TkrKalFitTrack contains a large number of methods aside from those defined in the interface (see following two slides). Can some of these be collected together/removed/etc.?
  – E.g. Can the calls to get the $\chi^2$ have a “type” argument?
Current Track Fit TDS Output Structure

/// Constructor/destructor for the class
TkrKalFitTrack();
TkrKalFitTrack();
enum Status {EMPTY, FOUND, CRACK};
void initializeBase(Status stat, int type, double energy0, Point& x0, Vector& dir);
void initializeQual(double chiSq, double ChiSqSmooth, double rms, double quality, double e, double e_err, double ms);
void initializeGaps(int xgaps, int ygaps, int x1st, int y1st);
void initializeKal(int nSegPoints, double chisqSeg, int nxHits, int nyHits, double radlen); // Define the TkrRecInfo access methods
/// Provides access to the basic information needed by external users
double getQuality() const {return m_Q;};
double getEnergy(TrackEnd end = Start) const { return getFoLPlane(end).getEnergy(); } }
int getLayer(TrackEnd end = Start) const { return getFoLPlane(end).getIDPlane(); }
int getTower(TrackEnd end = Start) const { return getFoLPlane(end).getIDTower(); }
Point getPosition(TrackEnd end = Start) const { return getFoLPlane(end).getPoint(TkrFitHit::SMOOTH); }
Vector getDirection(TrackEnd end = Start) const;
Ray getRay(TrackEnd end = Start) const;
TkrFitPar getTrackPar(TrackEnd end = Start) const { return getFoLPlane(end).getHit(TkrFitHit::SMOOTH).getPar(); }
double getTrackParZ(TrackEnd end = Start) const { return getFoLPlane(end).getZPlane(); }
TkrFitMatrix getTrackCov(TrackEnd end = Start) const { return getFoLPlane(end).getHit(TkrFitHit::SMOOTH).getCov(); }
bool empty(int numHits) const; // Utility void writeOut(MsgStream& log) const;
/// Access to primary quantities on track quality and scattering info
inline Status status() const {return m_status;};
inline double getChiSquare() const {return m_chisq;};
inline double getChiSquareSmooth() const {return m_chisqSmooth;};
inline double getScatter() const {return m_rmsResid;};
inline double getKalThetaMS() const {return m_KalThetaMS;};
inline double getKalEnergy() const {return m_KalEnergy;};
inline double getKalEnergyError() const {return m_KalEnergyErr;};
/// Access to derived information on gaps
int getNumGaps() const {return m_Xgaps + m_Ygaps;};
inline int getNumXGaps() const {return m_Xgaps;};
inline int getNumYGaps() const {return m_Ygaps;};
inline int getNumXFirstGaps () const {return m_XistGaps;};
inline int getNumYFirstGaps () const {return m_YistGaps;};
/// Access to fit specific information
inline Point getInitialPosition() const {return m_x0;};
inline Vector getInitialDirection() const {return m_dir;};
inline Point getPosAtZ(double deltaZ) const {return m_x0+deltaZ*m_dir;};
inline double getStartEnergy() const {return m_energy0;};

inline int getNumSegmentPoints() const {return m_numSegmentPoints;}
inline double chiSquareSegment(double penaltyGap = 0.) const {return m_chisqSegment + penaltyGap*getNumGaps();}
inline int getNumXHits() const {return m_nxHits;}
inline int getNumYHits() const {return m_nyHits;}
inline int getType() const {return m_type;}
inline double getTkrCalRadlen() const {return m_TkrCal_radlen;}

// Access to derived information on kinks
double getKink(int iplane) const;
double getKinkNorma(int iplane) const;

// Access to plane information
int getNumHits() const {return size();}
TkrFitPlane getFoLPlane(TrackEnd end = Start) const;

// Set functions for initializing the data members
inline void setInitialPosition(const Point x) {m_x0 = x;}
inline void setInitialDirection(const Vector x) {m_dir = x;}
inline void setChiSquare(double x) {m_chisq = x;}
inline void setChiSquareSmooth(double x) {m_chisqSmooth = x;}
inline void setChiSqSegment(double x) {m_chisqSegment = x;}
inline void setQuality(double x) {m_Q = x;}
inline void setScatter(double x) {m_rmsResid = x;}
inline void setKalThetaMS(double x) {m_KalThetaMS = x;}
inline void setKalEnergy(double x) {m_KalEnergy = x;}
inline void setKalEnergyError(double x) {m_KalEnergyErr = x;}
inline void setNumXGaps(int i) {m_Xgaps = i;}
inline void setNumYGaps(int i) {m_Ygaps = i;}
inline void setNumXFirstGaps(int i) {m_XistGaps = i;}
inline void setNumYFirstGaps(int i) {m_YistGaps = i;}
inline void setStartEnergy(double x) {m_energy0 = x;}
inline void setNumSegmentPoints(int i) {m_numSegmentPoints = i;}
inline void setNumXHits(int i) {m_nxHits = i;}
inline void setNumYHits(int i) {m_nyHits = i;}
inline void setType(int i) {m_type = i;}
inline void setTkrCalRadLen(double x) {m_TkrCal_radlen = x;}
inline void setStatus(Status status) {m_status = status;