A proposal for data format of track quality and track merging in CMSSW

Requirements

• No changes to the reco::Track class
• Associate track quality to reference to track
• Minimal size of data formats
• Being able to handle different sources of tracks (rs, ctf, gsf)
• Keep track of provenance of tracks
Proposal 1

- Each track reconstruction path removes fake to a certain level
  - xxFinalTracks (xx=ctf, rs, ...)

- The merger module *mergedTracks* runs on all those collections
  - produce one collection for each
    - *mergedTrack*
  - **Set a provenance “bit”** to the reco::Track

- The filter module *trackQualityProducer* runs this collection (*mergedTrack*)
  - **Set a quality “bit”** to the reco::Track

- Event Size:
  - just adding 2 uint on top of a lot of things <1% increase in size

- Pros:
  - Easy access in FWLite (track.quality()==tight, track.provenance()==CTF,...)

- Cons:
  - Provenance “bit” is a “waste of event size (not that much waste anyway)
  - Need to drop&create track collection to set quality “bit”
  - Loose specific track type (GsfTracks, DAFTracks,...)
Proposal 2

- Each track reconstruction path removes fake to a certain level
  → xxFinalTracks (xx=ctf, rs, ...)

- The merger module \textit{mergedTracks} runs on all those collections
  → produce a \textbf{separate clean} collection for each
    - \textit{mergedTrack:ctfFinalTracks}
    - \textit{mergedTrack:rsFinalTracks}
    - ...

- The filter module \textit{trackQualityProducer} runs on all those collections
  → Produces one single ValueMap<trackQuality> object
  → Provenance kept by the instance name and the RefToBase in ValueMap

- Event Size:
  → just adding 2 uint on top of a lot of things <1% increase in size

- Pros:
  → Full Provenance
  → RefToBase possible from ValueMap (CMSSW)
  → Minimal event size to keep track of provenance

- Cons:
  → ValueMap and FWLite ? See usage slide later on in FW.
Proposal 2

• \textit{reco::TrackQuality} holds the track quality status bit (DataFormats)

• Helpers that calculate \textit{TrackQuality} from Event and Track (tools)

• \textit{edm::ValueMap<reco::TrackQuality>} as the quality collection
  → Can hold “association” from different productID
  → ProductID is not stored many times (gain in size w.r. Ref)

• \textit{edm::ValueMap<T>::Parser<0>} as the Parser
  → ctor(const Event&, const ValueMap<T>&)
  → regular \texttt{begin()/end()} iterators
  → special \texttt{begin(const T&t)} and \texttt{next(const T&t)} members
namespace reco {
    typedef edm::ValueMap<reco::TrackQuality> TrackQualityValueMap;
}
namespace edm { namespace helper {
    typedef reco::TrackQualityValueMap::Parser<reco::Track> TrackQualityParser;
}}

edm::Handle<reco::TrackQualityValueMap> qualityMapH;
iEvent.getByLabel(theTag, qualityMapH);

edm::helper::TrackQualityParser parser(iEvent, *qualityMapH);
edm::helper::TrackQualityParser::iterator it = parser.begin();
edm::helper::TrackQualityParser::iterator end = parser.end();
for (; it!=end; ++it){
edm::helper::TrackQualityParser::iterator::value_type association = it.get();
edm::helper::TrackQualityParser::iterator::first_type ref = association.key;
edm::helper::TrackQualityParser::iterator::second_type quality = association.value;
const edm::Provenance & provenance = iEvent.getProvenance(ref.id());}
namespace reco {
    typedef edm::ValueMap<reco::TrackQuality> TrackQualityValueMap;
}
namespace edm { namespace helper {
    typedef reco::TrackQualityValueMap::Parser<reco::Track> TrackQualityParser;
}

edm::Handle<reco::TrackQualityValueMap> qualityMapH;
iEvent.getByLabel(theTag, qualityMapH);

edm::helper::TrackQualityParser parser(iEvent, *qualityMapH);
reco::TrackQuality aCertainQuality(reco::TrackQuality::TIGHT);
edm::helper::TrackQualityParser::iterator it = parser.begin(aCertainQuality);
edm::helper::TrackQualityParser::iterator end = parser.end();
for (; it!=end; it.next(aCertainQuality)){
edm::helper::TrackQualityParser::iterator::value_type association = it.get();
edm::helper::TrackQualityParser::iterator::first_type ref = association.key;
edm::helper::TrackQualityParser::iterator::second_type quality = association.value;
const edm::Provenance & provenance = iEvent.getProvenance(ref.id());}