Luminosity and FileSummaryRecords

Data flow
Offline work flow
Handling of Lumi data
Operations with FSR

Jaap Panman 19 June 2009

Questions

- What type of object is going in the FSR (e.g. counters, ratios, averages)?

- How are the objects accumulated (How are they identified, what operations are needed on them, and when (execute, finalize etc.))?

- In the job creating the objects, do they have to be accessed by algorithms other than the one accumulating them?

- Is there a need to propagate the objects to later processing steps (i.e. should they be copied to the output file if read from the input file)?

- Is there a need to combine objects from the FSR of several input files? If so, with what operations, and when?

Online

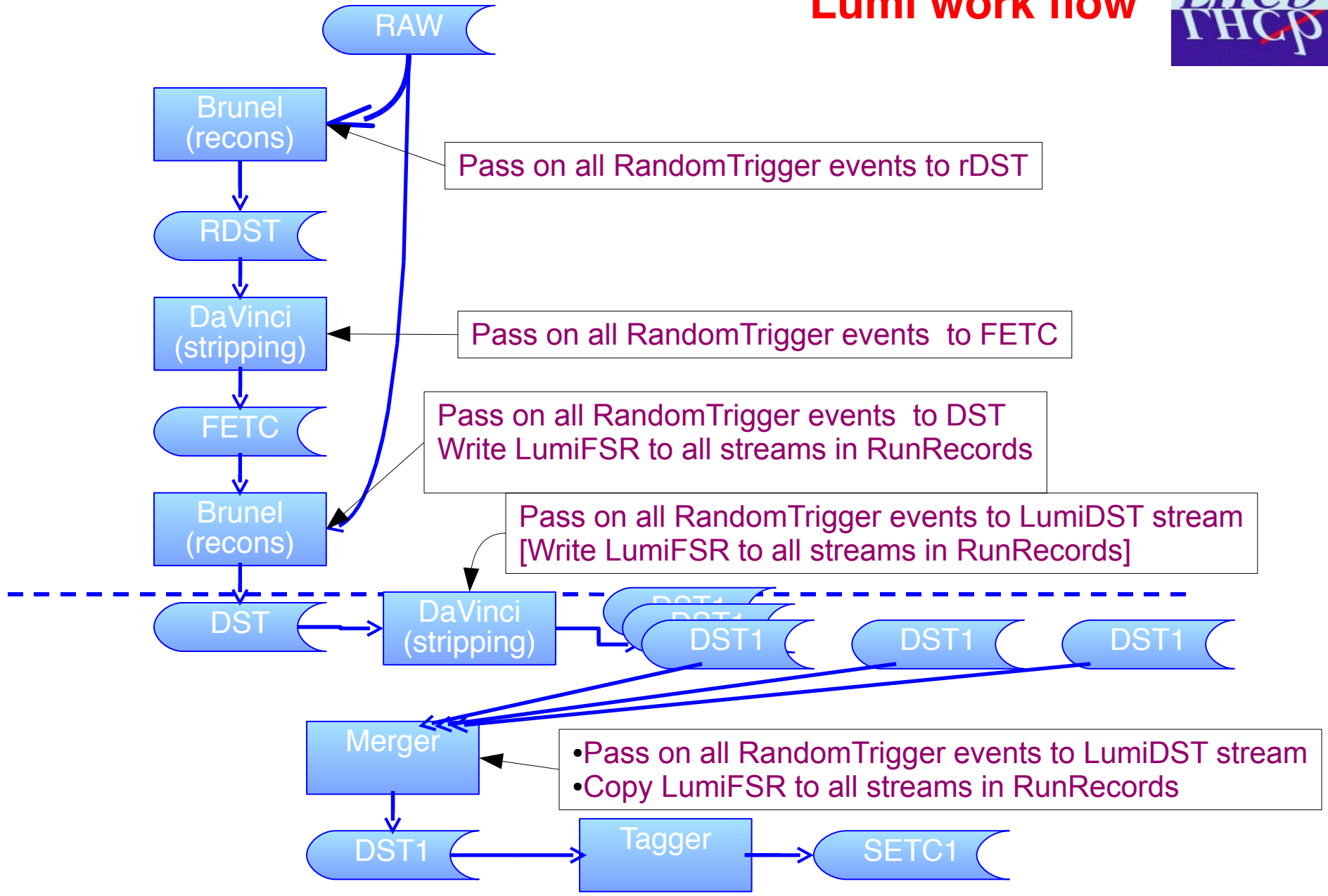
In MOORE

Pick up interesting quantities (various rates) in each `RandomTrigger` event
Transient class (TES) contains key-value pairs of these counters (`HltLumiSummary`)
Data are made persistent in `RAWDATA` bank using `HltLumiSummaryWriter`
If this was the only trigger, discard all other data

In Monitoring farm

Only look at `RandomTrigger` events
Decode `HltLumiSummary using HltLumiSummaryDecoder`
And make some plots

Lumi work flow



Offline 1

In Brunel (first pass)

No need to handle `RandomTrigger` events, just make sure they are passed on to the rDST (this is to ensure that they are on the FETC later)

In DaVinci (first pass)

Pass all `RandomTrigger` events to FETC

Offline Brunel II

In Brunel (second pass)

Only look at `RandomTrigger` events

Decode `HltLumiSummary using HltLumiSummaryDecoder, recreate (HltLumiSummary)`

Algorithm `LumiAccounting` creates class `LumiFSR`

`LumiFSR` contains:

- List of runs (should be only one)
- List of input fileIDs (should be only one – switch when new file encountered)
- Vector of pairs: Key-pair(counter – integral) for each counter seen
  - counter/integral are 64 bit integers to ensure range
- One FSR per BXType per inputFile ! Keep BXTypes separate.

`FSRWriter` is instantiated and makes this class persistent in the `FileRecords` tree under `LumiFSR`

Note 1: The nanoEvents have to be passed on to the DST unmodified

Note 2: It is assumed that the input file ID is the one of the original online MDF file to label the FSRs uniquely

Offline Davinci II

In DaVinci (second pass)

Algorithm `LumiIntegrateFSR:`
Define "primary" BXType, i.e. BeamCrossing which contains the "signal"
Look at all events, just count them and detect fileID change

At FileID change:
Search FSR tree recursively and look for all FSRs below the node with the present fileID
Subtract background of the non-primary Bxs from the primary, normalizing counter-by-counter to the primary.
Result in `LumiIntegral` class, which is similar in structure to `LumiFSR,` but stores the values in doubles.
This class is never persisted!

It is important to "touch" all FSRs to make sure they are present on the TES

`FSRWriter` is instantiated and makes this class persistent in the `FileRecords` tree under `LumiFSR,` but one level down in the tree.

Offline DaVinci II (2)

In DaVinci (second pass - 2)

The stripping should:

- Remove the nanoEvents from the output DST for physics streams
- Create a special stripping stream for all lumiEvents (nano and full)
- Pass on all FSRs

Offline analysis

In DaVinci (user pass - analysis)

Algorithm `LumiIntegrateFSR`: should be run if the user is interested in luminosity. It will do exactly the same as in the production pass

The Algorithm uses a tool `LumiIntegrator` to integrate the per-file-FSRs over the whole set of input files
The tool CAN inspect (in the future) a database with calibration constants per run and with absolute normalization factors to obtain the "integrated luminosity"

If the user addresses the same instance of the tool the result can be used to normalize e.g. Histograms at finalize.

Note 1: more than one instance of algorithm and tool can be created, so that BeamCrossing and B1Gas can be normalized independently.

Note2: The final integral is NEVER persisted! (except on the printout maybe)

Request

In DaVinci (user pass - analysis)

The integrate algorithm counts all events seen by the user per input file.

If there is an additional FSR which contains the total number of events written to the output file by the production step, the tool can check if the user has seen all events, i.e. If the normalization number is consistent.

So need an FSR with the events written by the production

Questions - answers

- What type of object is going in the FSR (e.g. counters, ratios, averages)?
  Integer counters

- How are the objects accumulated (How are they identified, what operations are needed on them, and when (execute, finalize etc.))?
  Addition in **execute**

- In the job creating the objects, do they have to be accessed by algorithms other than the one accumulating them?
  Tool

- Is there a need to propagate the objects to later processing steps (i.e. should they be copied to the output file if read from the input file)?
  Absolutely!

- Is there a need to combine objects from the FSR of several input files? If so, with what operations, and when?
  The FSR stays separate