Adding support for Random Access to LCIO files

Adding random access to LCIO files.

- Goals
- Proposed implementation
- Reading files with random access records.
    - Implementation notes
- Changes needed in SIO
- Changes needed in LCIO
- Adding support for accessing tagged events

Goals

Allow efficient access to specific events in LCIO files. Events should be selectable by

- Run
- Run+Event
- Index (i.e. 10000th event in file)
- Tag (e.g. EMISS>200)
- Access must work for "chains" of files in addition to individual files.

The last criteria is optional, since there is not an urgent requirement for it, but it would be useful to at least consider how this could be supported in future.

Proposed implementation

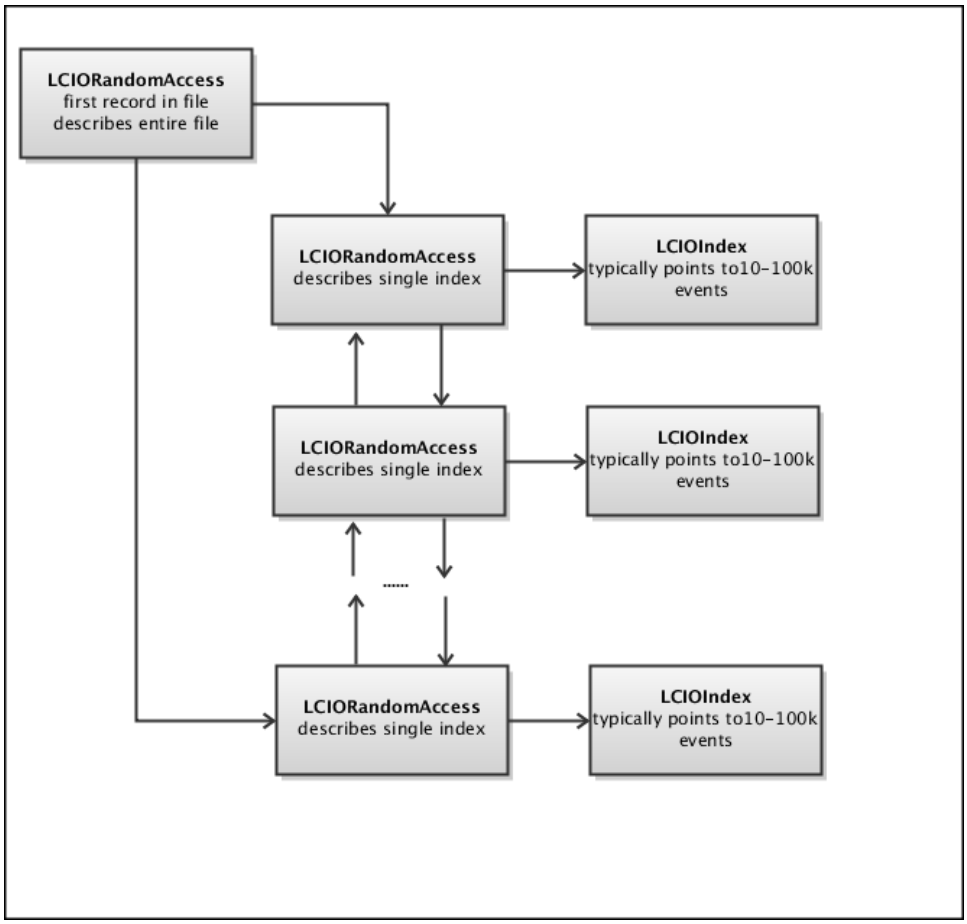
One way to implement this would be to (optionally) include two new types of records in LCIO files.

LCIORandomAccess.xml

```
<record name="LCIORandomAccess">
   There are two types of LCIORandomAccess records
     file record -- one per file, always first record on file
     index record -- one or more per file, points to associated LCIOIndex record
   <block name="LCIORandomAccess" major="1" minor="0">
       <data type="int" name="runMin"/>
       <data type="int" name="eventMin"/>
       <data type="int" name="runMax"/>
       <data type="int" name="eventMax"/>
       <data type="int" name="nRunHeaders"/>
       <data type="int" name="nEvents"/>
       <data type="int" name="recordsAreInOrder"/>
       <data type="long" name="indexLocation">
          Location in file of associated index. Always null for file record.
       </data>
       <data type="long" name="prevLocation">
          For file record location of first index record in file
          For index record location of previous index record (or null if first)
       </data>
       <data type="long" name="nextLocation">
          For file record location of last index record in file
          For index record location of next index record (or null if last)
       </data>
       <data type="long" name="firstRecordLocation">
           For index record location of the first record associated with this block
       </data>
   </block>
</record>
```

LCIOIndex.xml

```
<record name="LCIOIndex">
   <block name="LCIOIndex" major="1" minor="0">
       <data type="int" name="controlWord">
          Bit 0 = single Run
          Bit 1 = long offset required
       </data>
       <data type="int" name="runMin"/>
       <data type="long" name="baseOffset"/>
       <data type="int" name="size"/>
       <repeat count="size">
           <if condition="(controlWord&amp;amp;1)==0">
              <data type="int" name="runOffset">
                  Relative to runMin
              </data>
           </if>
           <data type="int" name="eventNumber">
              Event number, or -1 for run header records
           </data>
           <if condition="(controlWord&amp;amp;2)==1">
              <data type="long" name="locationOffset">
                  Relative to baseOffset
              </data>
           <else/>
              <data type="int" name="locationOffset">
                  Relative to baseOffset
              </data>
           </if>
       </repeat>
   </block>
</record>
```



If the file is written with support for random access enabled then the first record in the file will be a LCIORandomAccess record which describes the entire file. This block will then point to one or more LCIORandomAccess records elsewhere in the file, each of which will have an associated LCIOIndex record. The LCIOIndex record contains the location of each EventHeader or RunHeader record in the file.

Reading files with random access records.

The LCIO interface already contains methods such as:

- Read all RunHeaders
- Read all Events
- Skip N events
- Skip to Run n Event m

the initial goal would be to implement these methods more efficiently. In addition it may be useful to add some new methods, such as:

readStream(String criteria, int maxEvents)

which would read only those events satisfying some selection criteria (such as "EMISS>200").

Implementation notes

- The purpose of having two types of records is that the (small) LCIORandomAccess records can be written uncompressed. This allows them to be read quickly, and updated after they are written using the SIO random access mechanism described below. The larger LCIOIndex records can be written with compression turned on since they never need to be updated once they are written.
- The reason for having multiple LCIOIndex records in a file is to prevent the problem of having to store potentially infinitely large index blocks in memory. By storing something like 10-100k records per index block performance to access records is still good, without huge memory usage even for very large (chains) of files. This also makes the task of appending to an existing files much easier.
- Older LCIO readers should ignore/skip the LCIOIndex/LCIORandomAccess records.

Changes needed in SIO

Small changes are needed in the SIO library to support random access.

- Changes for writing files
  - When creating a record return the location in the file of the record

    - Allow existing records to be overridden. New record must be exactly the same size as existing record. The file format on disk is completely unchanged. Because size of compressed records cannot be predicted, replacing compressed records is not supported.
    - Optionally allow space to be reserved for a future record (not required for LCIO).
- Changes for reading
    - Allow record at a given location to be read.

A prototype updated version of the Java SIO library is available which incorporates these modifications. The documentation is here. Newly added methods (in SIOWriter and SIOReader) are marked **since 2.1**. The changes to support this were fairly minor and could presumably be added to the C++ SIO code in a similar way.

Changes needed in LCIO

A prototype of the Java code has been implemented in the LCIO repository on a branch called random_access_io_branch. The changed classes are all in package hep.lcio.implementation.sio. The changes are in:

- SIOLCWriter – modified to (optionally) write the new record types
- SIOLCReader – no changes to functionality, but refactored to make it easier to extend
- SIOLCRandomAccessReader – the main changes are here. This class extends SIOLCReader, and when reading files without the new records just delegates to the the super class. When reading files with the new records it provides efficient implementations of all of the LCIO methods including
- readEvent(int runNumber, int evtNumber)
- skipNEvents(int n)
- readNextRunHeader()

This efficient access works for single files and also for chains of files opened with

- open(String[] filenames)

The implementation of these methods is currently only intended as a proof of principle. If we decide to proceed we would need to clean up the code and add many test cases.

Adding support for accessing tagged events

The current implementation does not support accessing events by tags (e.g. EMISS>200) however this would be easy to add. LCIO already supports tagging events using parameter lists, which can include int, float or String values. For efficient access we could copy some or all of these tag values into the LCIOIndex block. This could be done something like this:

LCIOIndex.xml

```
<record name="LCIOIndex">
   <block name="LCIOIndex" major="1" minor="0">
       <data type="int" name="controlWord">
          Bit 0 = single Run
          Bit 1 = long offset required
          Bit 2 = LCParameters included
       </data>
       <data type="int" name="runMin"/>
       <data type="long" name="baseOffset"/>
       <data type="int" name="size"/>
       <if condition="(controlWord&amp;amp;2)==1">
          <data type="short" name="nIntParameters"/>
          <repeat count="nIntParameters">
              <data type="string" name="key">key (name) of parameter</data>
          </repeat>
          <data type="short" name="nFloatParameters"/>
          <repeat count="nFloatParameters">
              <data type="string" name="key">key (name) of parameter</data>
          </repeat>
          <data type="short" name="nStringParameters"/>
          <repeat count="nStringParameters">
              <data type="string" name="key">key (name) of parameter</data>
          </repeat>
       </if>
       <repeat count="size">
           <if condition="(controlWord&amp;amp;1)==0">
              <data type="int" name="runOffset">
                  Relative to runMin
              </data>
           </if>
           <data type="int" name="eventNumber">
              Event number, or -1 for run header records
           </data>
           <if condition="(controlWord&amp;amp;2)==1">
              <data type="long" name="locationOffset">
                  Relative to baseOffset
              </data>
           <else/>
              <data type="int" name="locationOffset">
                  Relative to baseOffset
              </data>
           </if>
           <if condition="(controlWord&amp;amp;2)==1">
              <repeat count="nIntParameters">
                  <data type="int" name="value"/>
              </repeat>
              <repeat count="nFloatParameters">
                  <data type="float" name="value"/>
              </repeat>
              <repeat count="nStringParameters">
                  <data type="string" name="value"/>
              </repeat>
           </if>
       </repeat>
   </block>
</record>
```

With this extra information in the LCIOIndex blocks events satisfying arbitrary criteria could be located simply by scanning the LCIOIndex blocks.