TSkim : a tool for skimming ROOT trees

David Chamont
École polytechnique, CNRS/IN2P3, Laboratoire Leprince-Ringuet, 91218 Palaiseau, France.

Abstract. Like many HEP researchers, the members of the Fermi collaboration have chosen to store their experiment data within ROOT trees. A frequent activity of such physicists is the tuning of selection criteria which define the events of interest, thus cutting and pruning the ROOT trees so to extract all the data linked to those specific physical events. It is rather straightforward to write a ROOT script to skim a single kind of data, for example the raw measurements of Fermi LAT detector. This proves to be trickier if one wants to process also some simulated or analysis data at the same time, because each kind of data is structured with its own rules for what concerns file names and sizes, tree names, identification of events, etc. TSkim has been designed to facilitate this task. Thanks to a user-defined configuration file which says where to find the run and event identifications in the different kind of trees, TSkim is able to collect all the tree elements which match a given ROOT cut. The tool will also help when loading the shared libraries which describe the experiment data, or when pruning the tree branches. Initially a pair of PERL and ROOT scripts, TSkim is today a fully compiled C++ application, enclosing our ROOT know-how and offering a panel of features going far beyond the original Fermi requirements. In this manuscript, we present TSkim concepts and key features, including a new kind of event list. Any collaboration using ROOT IO could profit from the use of this tool.

1. Introduction
Like many HEP researchers, the members of the Fermi collaboration [1] [2] have chosen to store their data within ROOT trees [3]. A frequent activity of physicists is the tuning of selection criteria which define the physical events of interest, thus cutting and pruning the ROOT trees in order to extract all the data linked to those specific events. The TSkim tool has been designed in order to facilitate this task. Initially a pair of PERL and ROOT scripts, TSkim is today a fully compiled C++ application, offering a panel of features going far beyond the original Fermi requirements. Below, we review how and why the tool was born, what services it is offering today, and the new developments which are under consideration.

2. Motivation and data model
ROOT trees have been designed in order to support the typical organisation of HEP data: a tree is a sequence of elements (ROOT calls them entries or events), with each element handling all the data linked to a single physical event (real or simulated). All elements/entries/events have the same internal structure: a collection of variables (ROOT calls them branches or leaves) corresponding to the different quantities describing the event. ROOT trees are optimized for the fast retrieval of a subset of such branches, for all events matching a given expression.
The ideal previous description assumes that all the data linked to a given event fit in a single entry, and that all the entries fit in a single tree, but those assumptions are obviously wrong:

- The trees are stored in files, which have a maximum size defined by the operating system. The huge amount of data of any real experiment requires thousands of files. When reading back data, ROOT is able to stack all the trees of consecutive events and make them appear as a single long tree, called a ROOT chain.

- Not all the data linked to a given event is available at the same time. For example, one process writes the raw measurements (data acquisition), another one transforms them later (reconstruction) as positions and energies, and a third process identifies the particles (analysis). ROOT does not support the addition of new branches to an existing tree. Users are expected to create several collections of trees, for the different types of data. When one wants to reconstitute some events, from several types of data, he must declare the different types of trees (or chains) as “friends”. Thus ROOT will read them in parallel.

- The requirements of data acquisition are not the same as the requirements of reconstruction or data analysis, and those different types of data are often under the control of different user groups. This naturally leads to different choices for what concerns the name of files, the number of trees in each kind of file, the size of files and the way they are cut up, the names of trees, etc. Ultimately, tree entries are not “aligned”: the nth entry in a given tree does not necessarily refer to the same physical event as the nth entry in another kind of tree. That’s where ROOT currently reaches its limits: the friend mechanism will not easily support such non-aligned data.

How can TSkim help, and match the relevant entries? It relies on an assumption about the data model: each kind of tree must include two branches, one for the run identification and one for the event identification, and each pair of run/event numbers must uniquely identify a single physical event. So, if a user defines a selection of events for a given kind of trees, TSkim can establish the list
of run/event numbers for this selection, and retrieve the corresponding entries in other types of trees. The figure 1 shows a simplified example, inspired from Fermi data:

- Raw measurements are collected in “Digi” trees, within “digi” files.
- Reconstructed data is collected in “Recon” trees, within “recon” files.
- Important event characteristics are collected in “MeritTuple” trees, within “merit” files, together with “jobinfo” trees.

In almost all of those trees, two branches are containing the run/event numbers, with eventually different names (they are called “EvtRun” and “EvtEventID” in merit files). Thanks to a user-defined meta-data configuration file, TSkim knows the names of those branches for all types of trees, and can retrieve all the pieces of data for the physical event 2-15, for example. For the case of trees which lack those run/event branches, such as “jobinfo” in the example, TSkim currently always keeps the whole tree. This works when those trees are small, but would be a problem with big trees, and we currently are studying a way to let the user decide which tree to keep and which tree to drop. Also, we are aware that not all experiments can put in their trees such branches dedicated to run/event numbers. Even more, some types of data definitively cannot be attached to a specific physical event. Such non-event data is generally small, but not less important than event-data, and we must investigate how to manage it when skimming our data files. Our favorite idea is the exploitation of timestamps, that we expect to find in most types of non-event data, but we have not investigated that idea much yet.

3. Current features and input/outputs of a TSkim job

TSkim is a command-line tool, with a small PERL 5 front-end and a compiled C++ backend which only rely on ROOT classes. The basic task of TSkim is to take input ROOT files, containing ROOT trees, and produce similar output files with a subset of branches and events. This is tuned by shell environment variables and textual configuration files. The figure 2 shows a rather simple and typical way to launch a job.

```
csh> setenv ROOTSYS /.../v5.18.00c/root
csh> setenv TS_META_DATA .../MetaData.txt
```
```
csh> setenv TS_INPUT_LIBRARY_LIST .../LibraryList.txt
```
```
csh> setenv TS_INPUT_FILE_LIST .../FileList.txt
```
```
csh> setenv TS_INPUT_BRANCH_LIST .../BranchList.txt
```
```
csh> setenv TS_INPUT_EVENT_LIST .../EventList.txt
```
```
csh> setenv TS_DATA_TYPES "digi:reco:merit:jobinfo"
csh> setenv TS_OUTPUT_DIR $PWD
csh> tskim ...
```

**Figure 2.** Job example (csh flavor)

TSkim tries to help with most auxiliary tasks involved in a skimming job. The figure 3 emphasizes the various inputs to TSkim and its outputs. Figure elements are detailed in the subsections below.
3.1. Meta-data configuration file
At the very bottom of the figure, the meta-data file defines for each kind of data: the tree name, the name and C++ kind of the top branch (the single main branch at the base of the tree), the name of the run and event identification branches, and the name of the eventual data definition library (see 3.4.). Such a file is generally prepared once and for all for a given experiment data, although a user can choose to duplicate and extend it with his personal definitions. The figure 4 shows a simple example of a meta-data file, consistent with the data model in figure 1.

```plaintext
#!/ SECTION MetaData

(merit.treeName) MeritTuple
(merit.runIdBranchName) EvtRun
(merit.eventIdBranchName) EvtEventId

(jobinfo.treeName) jobinfo

(digi.treeName) Digi
(digi.runIdBranchName) m_runId
(digi.eventIdBranchName) m_eventId
(digi.topBranchName) DigiEvent
(digi.topBranchType) DigiEvent
(digi.libName) libdigiRootData.so
...
```

**Figure 4.** Example of meta-data configuration
3.2. Input data files
Obviously, the main job inputs are the files of event data (on the left), organized as input chains, one chain for each kind of data. The list of those files should be provided in a dedicated configuration files (at the bottom). Any file access protocol which is supported by ROOT can be used in this file list, including xroot. Also, one can declare a user-defined command to be called for the automatic generation of the file list, which typically interrogates some experiment specific data catalog.

The order of events within each file has no importance. On the contrary, for each kind of data, the files must be globally ordered so all events in a given file must have ids lower than the events in following files. We study how to support unordered files, but it turns to be difficult to implement a scalable mechanism.

3.3. Output data files
The output skimmed files stay on the right of the figure. It is worth noticing that the non-event “jobinfo” data is segregated in a separated file, so as to not interfere with the management of the maximum size of the skimmed merit files.

Upon user request, instead of producing those output files, TSkim can write a special kind of ROOT file which only contains the file names, tree names, and entry ranks of the selected data. We call this a “composite event list” (CEL). Such a list can be reused as input for a subsequent TSkim job. This is how one can apply successive cuts on different types of trees, while never duplicating real data. Together with an efficient xrootd server, physicists can progressively refine and exchange their collections of events, without writing any byte of real data.

3.4. Data definition libraries
On one hand, one can expect that the typing information included in the ROOT file is enough for ROOT to be able to merge and skim trees in those files: this proves true with flat trees whose branches use predefined low level ROOT types. One the other hand, when dealing with complex trees, whose entries were mapped from a complex C++ type, we have encountered frequent and tricky problems, so that Fermi users took to linking the exact data definition library which has been used when originally generating the data.

That’s why TSkim lets the user declare (or not) a data definition library to be loaded for each kind of data to be skimmed. In the figure, those libraries are depicted in the upper left corner. The user can either explicitly list the absolute paths of the libraries, in a dedicated configuration file, or give the body file names in the meta-data file, and the list of top directories where all the libraries stay. In the later case, TSkim will automatically parse the directories and locate the libraries for you. Last but not least, one can declare a command to be called for the automatic generation of the library list. Fermi has such a command, able to extract their specific header object from a given input file, inspect its content, establish the Fermi code release used when generating this data, and finally establish the list of corresponding data definition libraries.

It's worthwhile to mention that the data definition libraries have been compiled versus a given ROOT release, and cannot be loaded within an application compiled versus another ROOT release. That’s why, when installing TSkim, the administrator must compile it several times, once for each ROOT release used for an experiment’s input data. Today, that’s the user responsibility to choose one release or the other when about to launch a TSkim job. We are investigating how to make the choice automatic.

3.5. Event list
Before skimming, TSkim must establish what events should be kept. The list of those events can be explicitly provided by the user, but that’s obviously not the only way to proceed. The most common use-case is rather to declare a selection predicate (thanks to the shell variable TS_TCUT), based on the branches of a given kind of data (specified with TS_TCUT_DATA_TYPE), and this cut will be
evaluated for each entry. The figure 5 is a modified version of the example already shown in figure 2, with a tcut instead of the explicit event list.

csh> setenv ROOTSYS /.../v5.18.00c/root
csh> setenv TS_META_DATA .../MetaData.txt
csh> setenv TS_INPUT_LIBRARY_LIST .../LibraryList.txt
csh> setenv TS_INPUT_FILE_LIST .../FileList.txt
csh> setenv TS_INPUT_BRANCH_LIST .../BranchList.txt
csh> setenv TS_TCUT_DATA_TYPE merit
csh> setenv TS_TCUT "TkMomentum>200"
csh> setenv TS_DATA_TYPES "digi:reco:merit:jobinfo"
csh> setenv TS_OUTPUT_DIR $PWD
csh> tskim ...

Figure 5. Event selection through a TCUT

Upon request, TSkim can write a textual output file with the list of selected run/event numbers, with the same syntax required for the input event lists, as shown in figure 6. That same file can be reused, eventually modified, as an input event list for a further skimming job.

Even further, TSkim can write a CEL (composite event list), which encapsulates both a file list and an event list into a special binary ROOT file, as already said in section 3.3.

#! SECTION Events
#! 2000 entries in original dataset.
#! 7 events after cut: ...
1 8
1 183
1 344
1 553
2 117
2 517
2 980

Figure 6. Example of textual event list

#! SECTION Branches
(merit) -*
(merit) +PtT*
(merit) +Cal*
(digi) +m_eventId
(digi) +m_runId
(digi) +m_acd
(digi) -m_cal
(digi) ...

Figure 7. Example of branch list

3.6. Branches configuration file
As for events, one can explicitly declare the subset of branches to be kept for each kind of tree. The syntax of those declarations is the same as used by ROOT, prefixed with the data type within parenthesis, as shown in figure 7.

3.7. Bonus features
On top of providing a consistent interface to cut, prune, skim and whatever kind of data filtering one can think of, TSkim has evolved to also support the merging of files, because it involves a similar infrastructure. As a result, today, if you want to merge some files, just define neither event list nor cut,
and TSkim will do the job, taking profit of special merging features of ROOT, and even fast merging features if applicable.

Actually, an additional benefit of TSkim is that it will select for you the fast mode if and only if this is applicable for a given job. It will also inspect any new input data file and check that we do not encounter conditions known to be pathological:

- Do we have new unexpected branches? Or branches which have disappeared?
- Are the run/event numbers compatible with Root indexes?
- Are the run/event numbers well sorted?

4. Limits and future work

Although TSkim can be considered almost mature, we still have many ideas of improvements and new features. In the subsections below, we comment on new features we consider important. It is also an opportunity to emphasize and discuss some of the current limits of the tool.

4.1. Handle new data models

As already discussed at the end of section 2, unless one skims a single kind of tree, it is mandatory to work with trees which have branches including the run and event identification numbers. Although it is a current practice to have such branches (close to the ROOT concepts of major/minor variables), we fear it could be a showstopper for new interested experiments, and investigating how to remove that constraint. One easy improvement would be to add an option to support implicitly aligned trees (a given event is associated to the same entry rank in various trees). For non-event trees, we will investigate the use of timestamp branches.

4.2. Better support for very simple use-cases

For use-cases such as skimming a few ROOT files with the same single tree within, the use of TSkim should prove as simple as the use of ROOT hadd. This especially means:

- Adding command-line options, so that the user can say anything on the command-line, and does not need to define shell environment variables (as it is the case now).
- Adding a user specific configuration file, where he can set his favorite options.
- Making TSkim able to guess what is usually defined in the meta-data file.

4.3. Better support for very heavy use-cases

Fermi users have reached some limits when dealing with many small files, or few very large files. We must probably drop the internal use of TEventList or TEntryList for something more scalable, such as our composite event list. We are also fighting to reduce the number of loops through all data files, which is not affordable when the files are very remote. Currently, we still need two loops: one for selection of events, and one for the extraction. The only way to go further seems to drop the use of TTree::Draw.

4.4. Manage Meta-Data evolution

Nowadays, HEP experiments typically produce data over several years, and this data structure hardly stays the same during all that period. Therefore we face tricky problems when some user involuntarily tries to skim data sets wherein such a structure change happens. In most cases, ROOT is not able to correctly deal with such changes. TSkim is already able to detect and warn about the appearance or disappearance of a branch. We must enlarge the set of changes we are able to detect, and eventually find a way to repair them when possible. Also, ROOT has recently announced a new schema evolution mechanism [4], and we should investigate how we could profit from this new ROOT mechanism, so that TSkim could also become a tool for massive data reformatting.
4.5. Deliver a library
Currently, TSkim is delivered as a command-line tool, although it includes many ROOT utilities which are certainly worth sharing with users. We plan to improve the interface and documentation of our internal C++ classes, and deliver a library. This would prove especially useful for the management of CELs, which could be used directly as an input in external applications.

5. Conclusion
Thanks to a meta-data configuration file, which gives the names of event identification branches within the different types of trees, the user can define an event selection for a given kind of tree and let TSkim deduce and extract the associated entries from all other types of trees. Another entries matching strategy, based on timestamps, is under investigation.

Because merging and reformatting of data are related tasks, TSkim has been extended over years to also support those use-cases. Its implementation includes all known ROOT tips and tricks. When applicable, the smartest ROOT features are used (fast merging) and TSkim checks for known ROOT pathologic situations.

Our tool is already a valuable help for the Fermi users. Hopefully, we have demonstrated that it fulfills requirements which also apply to many other HEP experiments. If your experiment stores its data as ROOT trees and files, welcome aboard. Your event data model will probably exhibit some specific characteristics we have not anticipated, but we are ready to help. Also, volunteers are welcome to join the development. Most information is available at the main TSkim web site [5].

References
[1] Fermi/LAT Collaboration: W. B. Atwood, et al. “The Large Area Telescope on the Fermi Gamma-ray Space Telescope Mission”, submitted to Astrophysical Journal, arXiv:0902.1089v1. See also http://www-glast.stanford.edu/.
[2] R. Dubois. “GLAST LAT Science Data Processing - Four Months Experience After Launch”, to be published in ASP Conference Series Vol. XXX, Astronomical Data Analysis Software and Systems XVIII (ADASS 2008).
[3] R. Brun and F. Rademakers. “ROOT - An Object Oriented Data Analysis Framework”, Proceedings of AIHENP’96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/.
[4] P. Canal. “ROOT: Support For Significant Evolutions of the User Data Model”, to be published in Journal of Physics: Conference Series, proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2009).
[5] TSkim, http://llr.in2p3.fr/trac/tskim