RECENT DEVELOPMENTS IN THE ROOT I/O

Ph. Canal[#], FNAL, Batavia, IL 60510, USA
René Brun, Fons Rademakers, CERN, Geneva, Switzerland

Abstract

Since version 3.05/02, the ROOT I/O System has gone through significant enhancements. In particular, the STL container I/O has been upgraded to support splitting, reading without existing libraries and using directly from TTreeFormula (TTree queries). This upgrade to the I/O system is such that it can be easily extended (even by the users) to support the splitting and querying of almost any collections. The ROOT TTree queries engine has also been enhanced in many ways including an increase performance, better support for array printing and histograming, addition of the ability to call any external C or C++ functions, etc. We improved the I/O support for classes not inheriting from TObject, including support for automatic schema evolution without using an explicit class version. ROOT now support generating files larger than 2Gb. We also added plug-ins for several of the mass storage servers (Castor, DCache, Chirp, etc.).

We will describe in details these new features and their implementation.

STATUS

The ROOT development release 4.01/02 was issued on September 24th, 2004. We expect to issue the production release of the 4/01 series in December 2004.

There have been many improvements made to ROOT since CHEP 2003. This paper will concentrate on the improvements made to the I/O and TTree sub-system. Some of the other improvements have been described in other papers of the conference CHEP 2004 [1]. This includes the description of XROOTD, a novel ROOT File server which main characteristics are high performance, reliability and scalability [2]. The authentication layer using by PROOF and ROOTD was also completely overhauled [3]. The graphical interface was greatly improved with the introduction of "Object Property Editors", including the TH1Editor, TH2Editor, and TGraphEditor and the introduction of many new widget classes [4]. A GUI builder is now available and we added a brand new GL viewer. There were also many additions to the mathematical and statistical libraries, including a new more efficient implementation of the Matrix package, new functions in the TMath namespace and a new Quadratic programming package.

[#]pcanal@fnal.gov

ROOT I/O

TFile and TDirectory

We introduced support for 64 bits integers on all platforms via the portable typedef Long64_t (and ULong64_t), which maps to long long on Unix, _int64 with VC++ (and there unsigned counterpart). This allowed us to implement support for file larger than 2Gb added in ROOT 4.00. Files smaller than 2Gb are still readable by older version of ROOT. We also were able to add support for TTree with more than 2**31 entries.

A new data type, Double32_t, is now available for the case where the calculation should be done in double precision but where the resulting precision is less than single precision (float). Double32_t is stored as a Double_t in memory but is stored as Float_t on disk. This allows for a much better compression of the data on disk. ROOT also provides support for automatic schema evolution to and from float and double. Note that too many read and write cycles could result in some loss of precision.

XML output format

An update to the I/O classes (TBuffer in particular) allowed the customization of the storage backend. It has been first used to implement an XML back-end. It will also be used for SQL support in TTrees.

XML files will allow the interchange of data with applications unable to read ROOT file directly.

Refer to Sergey Linev's presentation for more detail [5].

ROOT I/O History

- Version 2.25 and older:

Only hand coded and generated streamer functions are supported. The end users must add support for schema evolution by hand. Enabling the I/O for a users class requires adding the ClassDef, ClassImp macros, to inherit from TObject, and to generate the CINT Dictionary

- Version 2.26:

This version saw the introduction of a mechanism for automatic schema evolution. This mechanism uses the class TStreamerInfo (which reads the information available in the dictionary) to drive a generic I/O routine.

- Version 3.03/05:

The macro ClassDef and ClassImp are no longer necessary to enable the I/O for classes not inheriting from TObject. Any non-TObject class can now be saved inside a TTree or as part of a TObject-class.

- Version 4.00/00:

Classes, which do not inherit from TObject and do not have a ClassDef macro, now benefit for an "Automatic versioning". This allows for a fully automatic schema evolution support.

- Version 4.00/08:

Introduction of an interface so that object not deriving from TObject can be saved directly in TDirectory.

- Version 4.01/02

TTree were updated to support more than 2 billions entries and to be able to automatically load the branch containing an object referenced by a TRef.

Foreign Objects

Since ROOT 4.01/02, the only thing needed to store a object in a ROOT file, is to generate a dictionary for its class. For versioning, a default value is provided using a Checksum based on the type and name of the persistent data members. When this default versioning is used, TBuffer stored the value '0' where it would otherwise store a class version number. Following this '0', the Checksum is stored as an additional 4 bytes.

Using ClassDef still presents a few advantages. ClassDef provides an IsA function which speeds up considerably the access to the TClass for a given object. In addition, when using ClassDef the version number (2 bytes maximum) consumes less space on disk than the "0+checksum".

TDirectory now has a new, safer interface to store and retrieve object:

*ptrclass *ptr=…; directory->WriteObject(ptr,"name");*
*ptrclass *ptr; directory->GetObject("name",ptr);*

This new interface can be used both for TObject and non-TObject classes. When calling TDirectory::GetObject, the 2nd parameter (ptr in the example) will be set to 0 unless an object whose name is the 1st parameter exist and is of an appropriate type to be used with a pointer of the type of 'ptr'.

TClonesArray

TClonesArray is a ROOT specific collection of TObject that has been designed to optimize I/O operations. In particular it optimizes the number of calls to new and deletes by pre-allocating the memory needed for its content. TClonesArray were also (until the introduction of TVirtualCollectionProxy) the only collection where the content could be split in a TTree. This splitting improves the compressions factor and run time and allows a partial retrieval of the contents.

Similarly, the content of TClonesArray can be saved "member-wise". This means that the same data members of all the elements of the collections are stored consecutively. This improves compression (because the buffered data more homogeneous). This also improves run-time, by avoiding n-1 tests on the data type of the data members. TClonesArray was also the only collection that was recognized by TTree::Draw. TClonesArray also enable the possibility to retrieve the content of the object even without the original compiled code.

Old STL Container Support

In versions of ROOT older than 4.00/00, all STL collections were always stored object wise. The nesting of STL collections within other STL collections was extremely limited and any addition required extensive modification of rootcint. No splitting of the content was possible. All STL containers were stored using a generated function. The compiled version of these functions required for writing and also for reading

New Container Support

For the new support of STL collections, and a priori any user collections, we introduced a new abstract interface: TVirtualCollectionProxy. It should be possible to implement this abstract interface for almost any collections.

This new interface enabled us to implement code that allows:

- Splitting (for collection of homogenous objects)
- Use in Tree Query (with automatic looping)
- Member-wise streaming (as opposed to Object wise streaming)

The rewrite was also done such that we now support effortlessly any arbitrary nesting of STL containers. It also enables to implement the reading of STL containers without the original compiled code (Emulated mode).

As of ROOT 4.01/02 only std::vector has an implement of TVirtualCollectionProxy.

The early prototype and some of the fundamental concepts were provided by Victor Perevoztchikov, BNL.

STL Support

Each STL container instance now has an associated TClass object. This makes the STL container an integral part of the ROOT class information system.

There are several co-existing streaming implementations:

- Generated Streamer
- Template Proxy (e.g. TVectorProxy)
- Emulation Proxy (e.g. TEmulatedVectorProxy)

The generate streamers are used for object-wise streaming. They fully respect custom allocators and comparators and are easier to implement and have similar run-time cost as a templated implementation.

The templated proxies (e.g.. TVectorProxy) are used for splitting and member-wise streaming. They fully respect custom allocators and comparators.

The emulation proxies (e.g.. TEmulatedVectorProxy) are used for reading without a compiled version. They allow the easy sharing of ALL ROOT files that have no custom streamers.

Why not rely only on the Emulation Proxy

Since the emulation proxies are implementing the full reading of the STL containers, we first thought of only using those emulation proxies. However this proved to have several insurmountable problems.

Implementing an emulation proxy that needs to act on a "live STL object" requires a few tricks and assumptions. One such trick is to assume that the memory footprint of the STL container object is independent from the template parameter. However even if this assumption is true, it still lead to some inelegant and limiting code. For example the list proxy would need a series of list of increasing fixed size content (i.e. list<char[1024]>, list<char[2048]>) to encapsulate each of the possible user object size!. In addition it proved almost impossible to implement this emulation proxies such that they would respect the custom allocators and comparators.

In addition, a template base implementation of the proxy can be much faster and more memory efficient.

If the emulation layer does not need to touch actual "live STL objects", then their implementation can be greatly simplified for example by using alternative collections (of similar run-time characteristics).

Container I/O Implementation

The new container I/O implementation is based on the idea that any container can be summarized by the sequence of its content's addresses. In particular, TVirtualCollectionProxy requires the implementation of an "At" member function. This member function is called via the operator [] by the I/O sub-systems.

This implementation allowed us to make the I/O sub-system completely independent of the collection. It reduced code duplication in TStreamerInfo and had no run-time cost for TClonesArray (and C-style array).

On the other hand, the implementation of TVirtualCollectionProxy for containers with no random access iterator will probably need to cache the iterator.

To differentiate the Member-wise saving of a collection from the object-wise saving, we are using the highest bit of the class 'version number' that is already saved for each STL collection. An API will be provided to select whether to save member-wise or object-wise for each data member that are STL collections

TTREE

TRef autoload

We added an optional support for the auto-loading of branches referenced by a TRef object. This is implemented by generating one table of references to branches per entry. TRef::GetObject uses this table to find and load the branch containing the referenced object. To enable this feature, call*: tree->BranchRef();*

TTree::GetUserInfo

This is a new method that can be used to store along side the TTree object any user-defined objects that are not depending on the entry number, for examples: Luminosity, Calibrations, etc.

In memory circular buffer

A new method, TTree::SetCircular enables a circular buffers for memory resident Trees. Once the TTree reach the given amount of entries, before appending any new entries, it first drops the oldest entries.

Copying a TTree

There are several very flexible and simple tools to copy TTree and TChain objects allowing cut on:

- Number of entries
- Number of branches
- Selection of entries base on a Formula

In ROOT 4.00, we removed the requirement for the user to explicitly set the addresses for all the branches. For example to copy all the branches except for one, simply use:

```
tree->SetBranchStatus("br",kFALSE");
newtree=tree->CloneTree();
```

To copy only a portion of the entries using a cut use:

```
tree->CopyTree("fTracks.fPx<=1.2");
```

TTree Queries

There have been many improvements to the TTree query engine. In particular we implemented a "Boolean expression optimization" for the operators && and ||, where the right operands of the expressions are evaluated only if the left operands' value are not enough to know the complete result of the expressions.

A new histogram editor was introduced which allows the rebinning of histograms generated from a TTree using the full information from the TTree (instead of being limited to just the binned information stored in the histogram).

We made several enhancements to the output of TTree::Scan, including the ability to properly display array content and adding the possibility to customize the size of the column being displayed.

The TTree queries can now contains calls to free standing functions or class static member functions. These functions can be either compiled or interpreted. They need to have only numerical arguments and return a numerical type. For Example:

```
tree->Draw("TMath::Prob(var,5)");
```

TTreeFormula now treats any collection class that has a TVirtualCollectionProxy in the exact same way as a TClonesArray:

- Automatically loops over the elements
- Can access a specific element
- Synchronized with other collections and arrays in the formulas

Connecting several TTrees

There are currently two ways to extend a TTree. A TChain can be used to extend the TTree vertically by collating several TTrees together and make them look like a single long TTree with the same structure but with more entries.

TTree Friends can be use to extend the TTree horizontally by virtually adding more branches to the hierarchy. Prior to ROOT 4.00/08 the correlation made between TTree Friends was only based on the entry number. This represents a serious problem if the TTree

objects have two sequences of entries that are semantically different. In ROOT 4.01, the TTree Friends can now be connected using an index.

Let's take the example where we have two trees each with a column for the Run Number and a column for the Event Number. One of the TTree (the main tree) is well sorted by Run Number and Event Number while the second tree (the user tree) is shorter but is not sorted by run number and event number. If these two trees are connected using a friend relation without an index, this will result in the loading of entries in both trees that corresponds to two different events. However if we index the main tree and make it a friend of the user tree, then when the user call GetEntry on the user tree, before loading the corresponding branch in the main tree, the tree will lookup in the index table the correct entry to read.

The MakeClass Revolution

There are currently several Fast Analysis Frameworks provided in ROOT. These include:

TTree::Draw

- For Fast histograming
- Load branch on Demand
- Only simple expressions

MakeCode

- C-Style
- Flat representation of the tree
- Obsolete

MakeClass

- Flat representation of the tree
- Difficulties with variable size arrays
- Branch loaded explicitly

MakeSelector

- Proof Ready
- Flat representation of the tree
- Difficulties with variable size arrays
- Branch need to be loaded explicitly

We are now implementing an elegant replacement for MakeClass/MakeSelector. It is currently named MakeProxy. The generated selector creates a C++ context where the branch names (including periods) can be used as a C++ variable. This new environment also provide on demand loading of branches. It respects or recreates the original class structure. It provides array bound checks and uses the user's shared libraries when it is available

MakeProxy Examples

The first tool to use MakeProxy is TTree::Draw when it is requested to draw a script. This feature of TTree::Draw allow the user to use complex looping, to call to any C++ functions or member functions, to write any arbitrary C++ and still provide on-demand loading of the branches.

RDBMS

New RDBMS interface: Goals

In ROOT 4.02 we want to provide access to any RDBMS tables from TTree::Draw interface.

We also want allow the creation a Tree in split mode which would create on the fly the equivalent DBMS table and fill it. The resulting table would then be able to be processed by SQL directly. This new interface would use the normal I/O engine, including support for automatic schema evolution.

New RDBMS Interface

We currently have a prototype that allows the creation of a table from a simple TTree (branch with leaf list) and the reading of simple RDMBS table.

This prototype already has two backends to access the database. One is implemented via ROOT's net package, including the TSQLServer and TSQLRow classes. We also have an implementation using the RDBC for reading [8]. The near future extension of this prototype should include support for branches of objects. For this, we would need to implement a way to store and retrieve TStreamerInfo(s) and TProcessID(s) in the database. We will probably use SQL binary 'blob' to store non-split objects.

FUTURE PLANS FOR I/O AND TTREE

In the next few months we intend to implement member-wise storing for std::vector and to develop the TVirtualCollectionProxy for each of the STL containers.

We also want to add support for auto loading of TRef branches across trees, to enhance the TTree indexing scheme to nicely work for TChain objects, especially when they are used in TTree Friendship.

We also want to enhance TTree::Draw so that it can follow transparently any TRef and TRefArray

REFERENCES

[1] http://indico.cern.ch/materialDisplay.py?contribId=85&sessionId=6&materialId=slides&confId=0 slide 3

[2] Andrew Anushevsky. "The Next Generation Root File Server", Chep2004

[3] Gerardo Ganis, "Authentication/Security Services In The Root Framework" , CHEP 2004

[4] Ilka Antcheva "Guidelines For Developing A Good Graphical User Interface", Chep2004

[5] Sergey Linev, "Xml I/O In Root", CHEP 2004

[6] Fons Rademakers, "Global Distributed Parallel Analysis Using Proof And Alien", CHEP 2004

[7] Maarten Ballintijn, "Super Scaling Proof To Very Large Clusters", CHEP 2004

[8] http://carrot.cern.ch/~onuchin/RDBC