XML I/O in ROOT

S. Linev, R. Brun, H.G. Essel
CHEP 2004
Motivation

- XML is well structured and formatted text format
- Widely used as metadata (configuration, parameters, geometry) storage format
- Good candidate to be used as exchange format between different applications
- ROOT has powerful I/O system, but had no (at the beginning of 2004) support of xml
eXtensible Markup Language (XML)

- Tree like structure (not ROOT tree) of text tags
- Each tag opened should be closed
- Tag can include other tags, contain text, has attributes
- In addition: DTD, XSLT, XML schema, namespaces, ...

```xml
<?xml version="1.0"?>
<Example>
  <item1>item text</item1>
  <item2 id="001">
    <subitem>subitem text</subitem>
  </item2>
  <item3 ref="001"/>
</Example>
```

Example values:

```
Tree like structure (not ROOT tree) of text tags
Each tag opened should be closed
Tag can include other tags, contain text, has attributes
In addition: DTD, XSLT, XML schema, namespaces, ...
```
XML packages

C/C++ based XML packages:
- libxml (Gnome)  [http://xmlsoft.org](http://xmlsoft.org)
- Xerces-C++ (Apache)  [http://xml.apache.org/xerces-c/](http://xml.apache.org/xerces-c/)
- expat (Mozilla)  [http://expat.sourceforge.net](http://expat.sourceforge.net)

Benchmarks of XML packages:  
Implementation details

- Slight changes (generalization) of TFile & TDirectory classes interfaces
- Virtualization of TBuffer interface
  - => 3-5% speed penalty
- New TXMLBuffer class to convert basic data types to/from xml tags
- New TXMLEngine class to provide narrow interface to libxml2 library => possibility to use other xml packages
- New TXMLFile & TXMLKey classes for manipulation with xml files via common ROOT TFile interface
- Since May 2004 in ROOT distribution
Example of xml file

```xml
<?xml version="1.0"?>
<root setup="2xoo" ref="null">
  <XmlKey name="named" cycle="1">
    <Object class="TNamed">
      <TNamed version="1">
        < TObject fUniqueID="0" fBits="3000000"/>
        <fName str="ObjectName"/>
        <fTitle str="ObjectTitle"/>
      </TNamed>
    </Object>
  </XmlKey>
  <XmlKey name="box" cycle="1">
    <Object class="TBox">
      <TBox version="2">
        < TObject fUniqueID="0" fBits="3000000"/>
        <TAttLine version="1">
          <fLineColor v="1"/>
          <fLineStyle v="1"/>
          <fLineWidth v="1"/>
        </TAttLine>
        <TAttFill version="1">
          <fFillColor v="19"/>
          <fFillStyle v="1001"/>
        </TAttFill>
        <fX1 v="0.000000"/>
        <fY1 v="0.000000"/>
        <fX2 v="1.000000"/>
        <fY2 v="1.000000"/>
      </TBox>
    </Object>
  </XmlKey>
</root>
```

```c++
void example() {
  TNamed named("ObjectName");
  TBox box(0.0, 0.0, 1.0, 1.0);
  TFile* fxml = TFile::Open("example.xml");
  named.Write("named");
  box.Write("box");
  delete fxml;
}
```
Supported features

- “Transparent” usage of XML files via ROOT plug-in manager
- Except TTree, most of ROOT classes, including TClonesArray, can be stored
- Support of custom streamers in user classes
- Different layouts of xml files: class-specific and generic
- Optional usage of xml namespaces
Generic and class-specific layouts

```xml
<Object class="TBox">
   <Class name="TBox" version="2">
      <Member name="TObject">
         <Item name="Version" v="1"/>
         <Item name="UInt_t" v="0"/>
         <Item name="UInt_t" v="50331648"/>
      </Member>
      <Member name="TAttLine" version="1">
         <Member name="LineColor" v="1"/>
         <Member name="LineStyle" v="1"/>
         <Member name="LineWidth" v="1"/>
      </Member>
      <Member name="TAttFill" version="1">
         <Member name="FillColor" v="19"/>
         <Member name="FillStyle" v="1001"/>
      </Member>
      <Member name="fX1" v="0.000000"/>
      <Member name="fY1" v="0.000000"/>
      <Member name="fX2" v="1.000000"/>
      <Member name="fY2" v="1.000000"/>
   </Class>
</Object>

<Object class="TBox">
   <TBox xmlns:TBox="http://root.cern.ch/..." version="2">
      <TObject fUniqueID="0" fBits="3000000"/>
      <TAttLine xmlns:TAttLine="http://root.cern.ch/..." version="1">
         <TAttLine:LineColor v="1"/>
         <TAttLine:LineStyle v="1"/>
         <TAttLine:LineWidth v="1"/>
      </TAttLine>
      <TAttFill xmlns:TAttFill="http://root.cern.ch/..." version="1">
         <TAttFill:FillColor v="19"/>
         <TAttFill:FillStyle v="1001"/>
      </TAttFill>
      <TBox:XT1 v="0.000000"/>
      <TBox:YT1 v="0.000000"/>
      <TBox:XT2 v="1.000000"/>
      <TBox:YT2 v="1.000000"/>
   </TBox>
</Object>
```
Data exchange scenarios

1. Communication between two ROOT applications
2. Import data from existing user C++ program with non-ROOT classes
3. Export data from existing ROOT application to other C++ application
4. Export data from existing non-ROOT xml files
Data exchange between ROOT applications

- Allows viewing/editing exchanged data with standard xml tools
- Has no big advantage compared to standard ROOT file format
Import data from C++ program

Supported in user classes:
- basic data types, arrays, const char*
- access to private and protected members
- objects, object pointers, array of objects, array of objects pointers
- classes inheritance tree
- STL string, vector, list, deque, set, map, multimap
Import data from C++ program

- Create ROOT dictionary for user classes
- Using **TXMLPlayer**, create xml-streamers for user classes
- Add xml-streamers and **TXmlFile** class to user project
- Both reading & writing of xml files becomes possible
- “ROOT-like” interface in user code to access xml files:

```cpp
TXmlFile file("test.xml");
TXmlEx1* ex1 = (TXmlEx1*) file.Get("ex1", TXmlEx1_streamer);
TXmlEx2* ex2 = (TXmlEx2*) file.Get("ex2", TXmlEx2_streamer);

TXmlFile outfile("test2.xml", "recreate");
outfile.Write(ex1, "ex1", TXmlEx1_streamer);
outfile.Write(ex2, "ex2", TXmlEx2_streamer);
```
Export data to C++ program

What is possible:
- do not use any ROOT classes, but then it is previous scenario
- generate class definitions and streamers for C++ code,
  converting ROOT containers to STL containers
Implementation is not yet clear and highly depend from real user requirements. Any suggestions are welcome.
Conclusion

- Generic XML I/O introduced in ROOT
- Several xml-layout options are supported
- Possibility for data exchange with other non-ROOT application

Investigation required:
- support of TTree?
- support for SAX interface to read fairly big files?
- use of alternative xml-engines?
- DTD-generation
- improvements in data exchange