Possibility of XML I/O support in ROOT

S. Linev, GSI, Darmstadt

ROOT Workshop 2004

Content

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

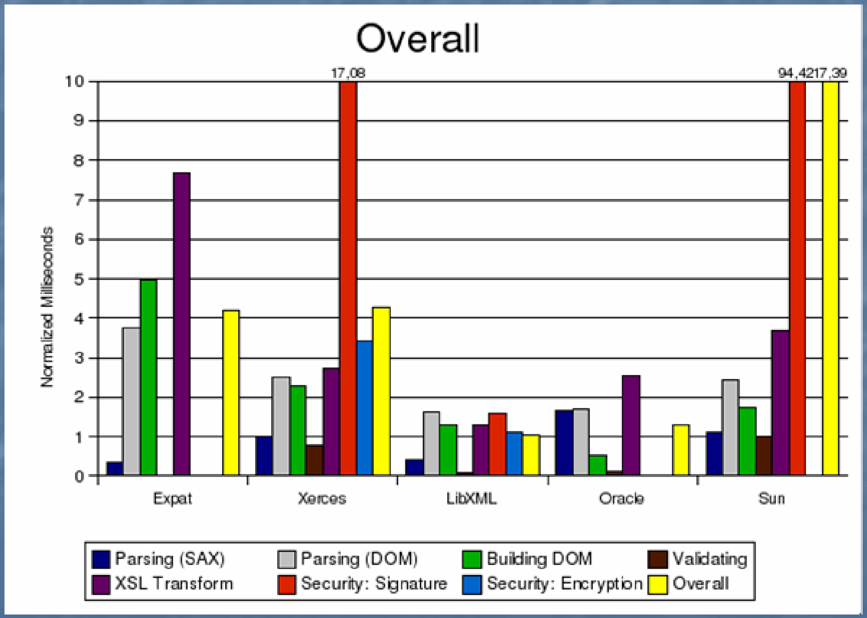
```
Example                  values:
 ├── item1                 “item text“
 ├── item2                 id="001"
 │    └── subitem          “subitem text“
 │
 └── item3                 ref="001"
```

XML packages

C/C++ based XML packages:

- libxml (Gnome) http://xmlsoft.org
- Xerces-C++ (Apache) http://xml.apache.org/xerces-C/
- expat (Mozilla) http://expat.sourceforge.net

Benchmarks of XML packages:
http://xmlbench.sourceforge.net



Usage of libxml2 library

Example of code to create XML file:

```
xmlDocPtr fDoc = xmlNewDoc(0);
xmlNodePtr fNode = xmlNewDocNode(fDoc, 0, (const xmlChar*) "Example", 0);
xmlDocSetRootElement(fDoc, fNode);
xmlNewTextChild(fNode, 0, (const xmlChar*) "item1",(const xmlChar*) "item text");
xmlNodePtr sub2 = xmlAddChild(fNode, xmlNewNode(0, (const xmlChar*) "item2"));
xmlNewTextChild(sub2, 0, (const xmlChar*) "subitem", (const xmlChar*) "subitem text");
xmlNewProp(sub2, (const xmlChar*) "id", (const xmlChar*) "001");
xmlNodePtr sub3 = xmlAddChild(fNode, xmlNewNode(0, (const xmlChar*) "item3"));
xmlNewProp(sub3, (const xmlChar*) "ref", (const xmlChar*) "001");
xmlSaveFormatFile("Example.xml", fDoc, 1);
xmlFreeDoc(fDoc);
```

XML and ROOT

- XML as metadata storage place: configuration, parameters and geometry objects
- XML files can be viewed and edited (with some restriction) with standard XML tools
- Data exchange between different packages

But currently:

- There is no XML support in ROOT (yet)
- Each new class requires its own XML streamer

Motivation

- ROOT has all class information in TStreamerInfo class with methods to serialize/deserialize objects
- Why not implement similar mechanism for XML, not only for binary ROOT format?
- Aim – introduce XML I/O in ROOT, where user should not write I/O code himself

Object representation in XML

```
class TXmlExample {
   public:
      Int_t          fValue;
      Double_t       fArray[5];
      TString        fStr;
      TXmlExample*   fSelfPtr;
   ClassDef(TXmlExample, 1);
};
```

object id

class name

basic type

array

string (special case)

pointer

```
<TXmlExample ref="id0">
   <fValue v="10"/>
   <fArray>
      <Double v="1.0"/>
      <Double v="10.0"/>
      <Double v="5.0"/>
      <Double v="7.0"/>
      <Double v="2.0"/>
   </fArray>
   <fStr TString="Hello"/>
   <fSelfPtr ptr="id0"/>
</TXmlExample>
```

First Implementation

- New class with two functions similar to TStreamerInfo::WriteBuffer() and TStreamerInfo::ReadBuffer()
  were implemented to serialize/deserialize objects to/from XML structures
- Libxml2 library was used
- Requires no any ROOT modifications

Problems

- Only relatively “simple” objects can be stored
- Custom streamers are not supported
- As a result, ROOT classes like histograms (TH1), containers (TObjArray) and many other can not be supported

TBuffer class modification

- Make six methods of TBuffer virtual:

```
void      WriteObject(const void *actualObjStart, TClass *actualClass);
void*     ReadObjectAny(const TClass* cast);
Int_t     CheckByteCount(UInt_t startpos, UInt_t bcnt, const TClass *clss);
void      SetByteCount(UInt_t cntpos, Bool_t packInVersion = kFALSE);
Version_t ReadVersion(UInt_t *start = 0, UInt_t *bcnt = 0);
UInt_t    WriteVersion(const TClass *cl, Bool_t useBcnt = kFALSE);
```

- Redefine these methods in new TXmlBuffer class to perform XML specific actions
- To support "TFile-like" key organization, new TXmlFile and TXmlKey classes have been created

Example with TObjArray

```
<?xml version="1.0"?>
<root>
 <XmlKey name="array" setup="1xxox">
  <TObjArray version="3">
   <XmlObject>
    <TNamed>
     <fName TString="name1"/>
     <fTitle TString="title1"/>
    </TNamed>
   </XmlObject>
   <XmlObject>
    <TNamed>
     <fName TString="name2"/>
     <fTitle TString="title2"/>
    </TNamed>
   </XmlObject>
   <XmlObject>
    <TNamed>
     <fName TString="name3"/>
     <fTitle TString="title3"/>
    </TNamed>
   </XmlObject>
   <XmlBlock size="9">
     00 00 00 00 03 00 00 00 00
   </XmlBlock>
  </TObjArray>
  <XmlClasses>
   <TNamed version="1"/>
  </XmlClasses>
 </XmlKey>
</root>
```

XmlKey with name and setup info

TObjArray tag with version

```
TObjArray arr;
arr.Add(new TNamed("name1", "title1"));
arr.Add(new TNamed("name2", "title2"));
arr.Add(new TNamed("name3", "title3"));
TXmlFile file("test.xml","1xxox");
file.Write(&arr, "array");
```

Part of TObjArray streamer (writing):

```
…
fName.Streamer(b);
nobjects = GetAbsLast()+1;
b << nobjects;
b << fLowerBound;
…
```

Now only version,
later full class info

```
<?xml version="1.0"?>
<root>
 <XmlKey name="array" setup="1xxox">
  <TObjArray version="3">
   <UChar>0</UChar>
   <Int>3</Int>
   <Int>0</Int>
   <XmlObject>
    <TNamed>
     <fName TString="name1"/>
     <fTitle TString="title1"/>
    </TNamed>
   </XmlObject>
   <XmlObject>
    <TNamed>
     <fName TString="name2"/>
     <fTitle TString="title2"/>
    </TNamed>
   </XmlObject>
   <XmlObject>
    <TNamed>
     <fName TString="name3"/>
     <fTitle TString="title3"/>
    </TNamed>
   </XmlObject>
  </TObjArray>
  <XmlClasses>
   <TNamed version="1"/>
  </XmlClasses>
 </XmlKey>
</root>
```

Consequence of TBuffer modification

- Most of ROOT classes can be stored
- Users classes with custom streamers can be supported
- Works, if reading and writing parts of custom streamer have similar sequence of I/O actions (normal situation)
- Some classes like TTree & TClonesArray are not tested and may be not required to be stored in XML format
- At worse case 10% lost of I/O performance

Still not fully acceptable because:

- this is just “hacking” of ROOT code
- TXmlFile and TXmlKey repeats a lot of functionality of similar TFile and TKey classes

Further investigations

- Producing of DTD files for validation purposes
- Using of XML namespaces to avoid names intersection
- Extension of TFile and TKey logic on XML files (via abstract interfaces)
- C++ code generator for XML I/O to access ROOT objects outside a ROOT environment
- Support of different XML packages

Conclusion

- There is no general XML I/O in ROOT
- Very limited solution possible without ROOT changing
- With slight TBuffer modifications acceptable XML support in ROOT is possible
- Further investigations required