New Xml Converters

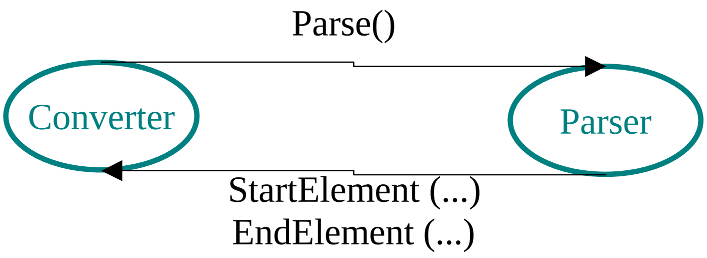
- General presentation of Xml converters
- The old way
  - SAX interface
  - Consequences on efficiency
- The new way
  - DOM interface
  - What we gain
- How to write a converter
  - Overview (general case and specific detector element case)
  - Real life examples
- References and documentation

Overview of Xml Converters

- One converter per object type
  - DetElem
  - LogVol
  - Isotope
  - MuonStation
  - VertexDetector
  - ...
- 4 main methods in IConverter interface to be implemented
  - createObj, updateObj, createRef, updateRef
  - Only createObj is actually implemented
- An underlying XML parser is used, namely xerces C^{++}
- The actual code does a (quasi) 1 to 1 mapping between XML elements and C^{++} objects and between XML attributes and C^{++} object members.

The SAX Interface (1)

- SAX is an interface to the XML parser based on streaming and call-backs
- You first need to implement the HandlerBase interface :
  - startDocument, endDocument
  - startElement, endElement
  - characters
  - warning, error, fatalError
- You should then give a pointer to your interface to the parser
- Then you call parse



Parse()

Converter

Parser

StartElement (...)

EndElement (...)

The SAX Interface (2)

```
<A>
    <B1>
        <C/>
    </B1>
    blabla
    <B2/>
</A>
```

```
StartDocument()
    startElement(A)
        startElement(B1)
            startElement(C)
            endElement(C)
        endElement(B1)
        characters("blabla")
        startElement(B2)
        endElement(B2)
    endElement(A)
endDocument()
```

XML File ⟷ **SAX calls**

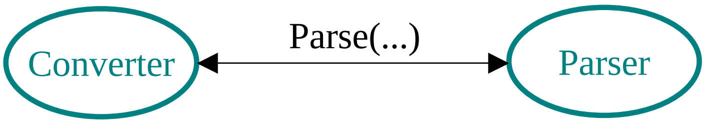
SAX pro and contra

- CONTRA
  - The file has to be parsed entirely to access any node. Thus, getting the 10 nodes included in a catalog ended up in parsing 10 times the same file.
  - Poor navigation abilities : no way to get easily the children of a given node or the list of "B" nodes
  - Made converters difficult to implement since the state of the parsing had to be handled by the user

- PRO
  - Low memory needs since the XML file is never entirely in memory
  - Can deal with XML streams

The DOM Interface (1)

- DOM is an interface to the XML parser based on tree representation of XML files
- One single method to parse files : parse. It returns a DOM_Document, the top node of the tree representing your file
- This tree is essentially made of :
  - DOM_Element : the xml tags
  - DOM_Attr : the xml attributes
  - DOM_Text : the bunches of text in XML
- You can navigate the tree with :
  - getAttribute, getAttributeNode, getAttributes
  - getChildNodes, getFirstChild, getLastChild, getParentNode
  - getNodeName, getNodeValue
  - GetElementsByTagName, getElementById

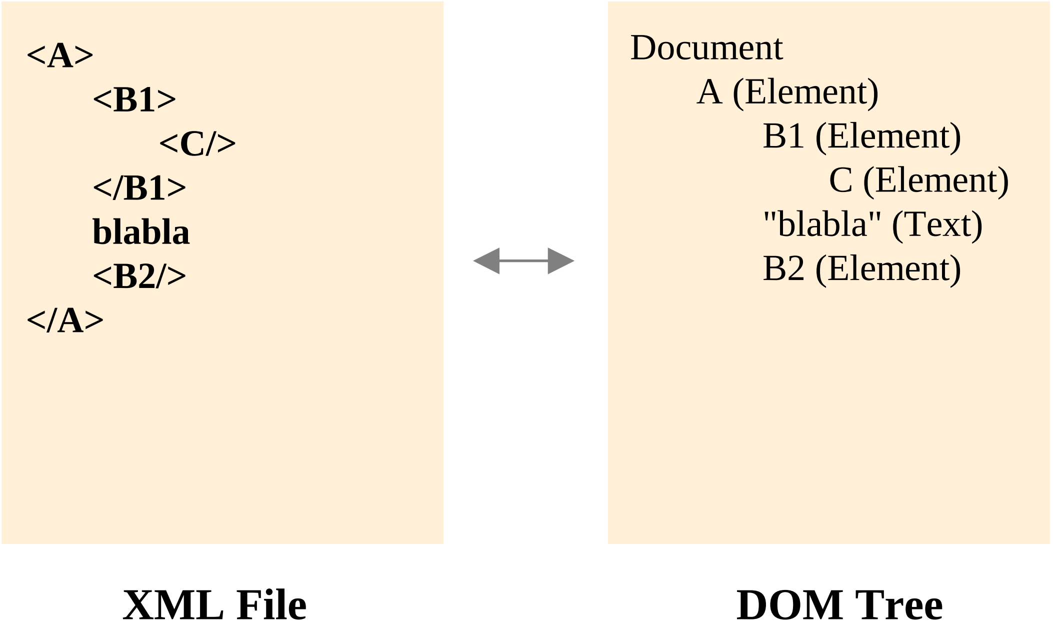


Converter

Parse(...)

Parser

The DOM Interface (2)



```
<A>
      <B1>
            <C/>
      </B1>
      blabla
      <B2/>
</A>
```

```
Document
      A (Element)
            B1 (Element)
                  C (Element)
            "blabla" (Text)
            B2 (Element)
```

XML File

DOM Tree

DOM pro and contra

- PRO
  - The file is parsed only once if you cache the DOM_Documents. A XMLParserSvc was created to encapsulate parsing and caching.
  - Still better, the file is not fully parsed if not necessary due to parse on demand implementation in the xerces parser.
  - High navigation abilities : this is the aim of the DOM design
  - Converters implementation very natural. No more state.
- CONTRA
  - More memory needed since the XML tree is in memory

Writing a converter

- XmlGenericCnv implements the whole machinery of looking for files, parsing them and getting the right DOM_Element from the tree.
- By inheriting from it, you only need to implement 4 methods :
  - i_createObj (DOM_Element, DataObject*&) : creation of the C^{++} object (new)
  - i_fillObj (DOM_Element, DataObject*) : called for each child of the DOM_Element that is also a DOM_Element
  - i_fillObj (DOM_Text, DataObject*) : called for each child of the DOM_Element that is a DOM_Text
  - i_processObj (DataObject*) : for computation can be made
- In addition one should use dom2Std to convert DOM_String to std::string. DOM_String::transcode() converts DOM_String ot char* but allocates memory
- XmlGenericCnv provides you the member xmlSvc that provides you an expression evaluator

XmlSurfaceCnv (1)

```
// Instantiation of a static factory class used by clients to create instances of this service
static CnvFactory<XmlSurfaceCnv>         s_FactoryXmlSurfaceCnv;
const ICnvFactory& XmlSurfaceCnvFactory = s_FactoryXmlSurfaceCnv;

// Empty Constructor
XmlSurfaceCnv::XmlSurfaceCnv (ISvcLocator* svc) : XmlGenericCnv (svc, classID()) {};


StatusCode XmlSurfaceCnv::i_createObj (DOM_Element element, DataObject*& refpObject) {

  // Object creation
  std::string elementName = dom2Std (element.getAttribute ("name"));
  Surface* dataObj= new Surface (elementName);
  refpObject = dataObj;

  // model attribute
  const std::string value = dom2Std (element.getAttribute ("model"));
  const double v_value = xmlSvc()->eval(value, false);
  dataObj->setModel (v_value);

  ...
} // end i_createObj
```

XmlSurfaceCnv (2)

```
StatusCode XmlSurfaceCnv::i_fillObj (DOM_Element childElement, DataObject* refpObject) {

   // gets the object
  Surface* dataObj = dynamic_cast<Surface*> (refpObject);

  // gets the element's name
  std::string tagName = dom2Std (childElement.getNodeName());

  // dispatches, based on the name
  if ("tabprops" == tagName) {
    const std::string address = dom2Std (childElement.getAttribute ("address"));
    long linkID = dataObj->addLink(address, 0);
    ...
  } else {
  ...
  }
}
```

Writing a specific DetElem Converter

- Detector elements can be extended by users (tag <specific>)
- To minimize the work, a templated class called XmlUserDetElemCnv<aType> has been created. It implements the whole conversion of a regular detector element.
- By inheriting from it, you only need to implement 1 method :
  - i_fillSpecificObj (DOM_Element, aType*) : called for each child of the <specific> tag that is also a DOM_Element

XmlMuonStationCnv

```
// Instantiation of a static factory class used by clients to create instances of this service
Static CnvFactory<XmlMuonStationCnv> muonst_factory;
const ICnvFactory& XmlMuonStationCnvFactory = muonst_factory;

// Empty Constructor
XmlMuonStationCnv::XmlMuonStationCnv(ISvcLocator* svc) :
  XmlUserDetElemCnv<DeMuonStation> (svc) {}

StatusCode XmlMuonStationCnv::i_fillSpecificObj (DOM_Element childElement,
                                                 DeMuonStation* dataObj) {
  // gets the element's name
  std::string tagName = dom2Std (childElement.getNodeName());

  if ("Al_plate_thickness" == tagName) {

    // get a value of the 'value' attribute
    const std::string value = dom2Std (childElement.getAttribute ("value"));
    if (!value.empty()) {
      dataObj->setThickness (xmlSvc()->eval(value));
    }
  } else {
  ...
  }
}
```

Documentation

- This presentation
- The xerces API (http://xml.apache.org/xerces-c/apiDocs/index.xml)
- The Gaudi documentation :http://proj-gaudi.web.cern.ch/proj-gaudi/Doxygen/v7/doc/html/index.html and http://lhcbsoft.web.cern.ch/LHCbSoft/LHCb/v7/doc/html/index.html
- The Ex/DetDescExample package where you can find some user specific detector element converters.