atlantis.event Package

July 16, 2003

1 Overview

The atlantis.event package deals with reading in of events from an XML file. Classes to represent each type of data from the ATLAS detector are contained in this package as well as a class to contain and manage the complete event.

Data files can be of a user defined type or the standard atlantis type. Events can also be read in from within a Zip file, however these events must be of the standard atlantis type.

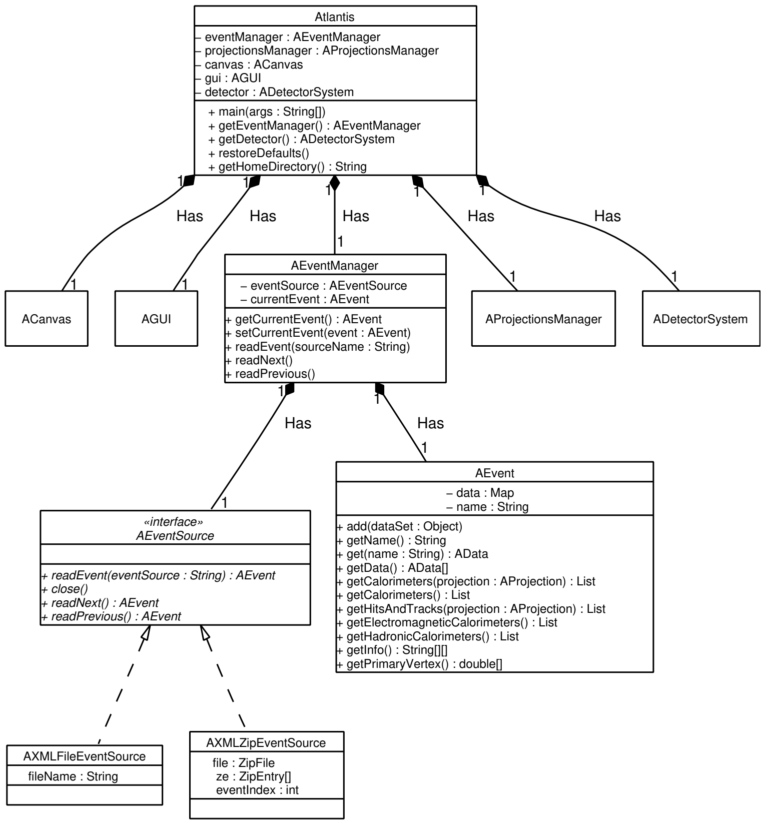

Figure 1: Event package - main classes

2 XML

XML is a text-based markup language, where data is identified using tags.

http://www.w3schools.com/xml/xml_whatis.asp

A Document Type Definition (DTD) specifies the kinds of tags that can be included in your XML document, and the valid arrangement of those tags. DTD can be used to ensure you don't create an invalid XML structure, and can be used to ensure the XML structure you are reading is valid. DTD specification is part of the XML specification rather than a separate entity, and it is optional.

http://www.w3schools.com/dtd/default.asp

3 Accessing XML

The current APIs for accessing XML documents either serially or in random access mode are respectively SAX and DOM.

Event XML files are read in using SAX - Simple API for XML. This is an Event based API, which reports parsing events whenever it sees a new tag directly to the application through callbacks. The application implements handlers to deal with different events.

The other type of XML API is DOM - Document Object Model. This is a tree based model that maps an XML document into an internal tree structure, thus enabling an application to navigate through the tree. Using this mechanism you can visit any part of the data at any time. DOM is used by Atlantis for the config.xml file at start up (this defines the User Interface) and also to read in the detector geometry information.

DOM is an easier API to use, but it requires reading the entire XML structure and holding the object tree in memory, so it is a lot more CPU and memory intensive than SAX.

4 SAX

Basic outline of SAX parsing - from java.sun.com

To start the process, an instance of the SAXParserFactory class is used to generate an instance of the parser.

The parser wraps a SAXReader object. When the parser's parse() method is invoked, the reader invokes one of several callback methods implemented in the application. Those methods are defined by the interfaces ContentHandler, ErrorHandler, DTDHandler, and EntityResolver.

Here is a summary of the key SAX APIs:

SAXParserFactory A SAXParserFactory object creates an instance of the parser determined by the system property, javax.xml.parsers.SAXParserFactory.

SAXParser The SAXParser interface defines several kinds of parse() methods. In general, you pass an XML data source and a DefaultHandler object to the parser, which processes the XML and invokes the appropriate methods in the handler object.

SAXReader The SAXParser wraps a SAXReader. Typically, you don't care about that, but every once in a while you need to get hold of it using SAXParser's getXMLReader(), so you can configure it. It is the SAXReader which carries on the conversation with the SAX event handlers you define.

DefaultHandler The DefaultHandler implements the ContentHandler, ErrorHandler, DTD-Handler, and EntityResolver interfaces (with null methods), so you can override only the ones you're interested in.

ContentHandler Methods like startDocument, endDocument, startElement, and endElement are invoked when an XML tag is recognized. This interface also defines methods characters and processingInstruction, which are invoked when the parser encounters the text in an XML element or an inline processing instruction, respectively.

ErrorHandler Methods error, fatalError, and warning are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). That's one reason you need to know something about the SAX parser, even if you are using the DOM. Sometimes, the application may be able to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, you'll need to supply your own error handler to the parser.

DTDHandler Defines methods you will generally never be called upon to use. Used when processing a DTD to recognise and act on declarations for an unparsed entity.

EntityResolver The resolveEntity method is invoked when the parser must identify data identified by a URI(Uniform Resource Identifier). In most cases, a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN(Uniform Resource Name)–a public identifier, or name, that is unique in the Web space. The public identifier may be specified in addition to the URL. The EntityResolver can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists.

A typical application implements most of the ContentHandler methods, at a minimum. Since the default implementations of the interfaces ignore all inputs except for fatal errors, a robust implementation may want to implement the ErrorHandler methods, as well.

5 Classes

2 Methods called from Atlantis.java main method:

- AEventManager.AEventManager();
- AEventManager.readEvent(String fileName);

5.1 AEventManager Class

Attributes :

- currentEvent:AEvent

- eventSource:AEventSource

Methods:

Constructor Sets the attributes to null.

readEvent(String sourceName) If the source name ends in .xml then an AXMLFileEventSource object is created. If it ends in .zip an AXMLZipEventSource object is created. These are cast to AEventSource. Then the AEventSource.readEvent(sourceName) method is called which returns an AEvent object. The currentEvent Attribute is then set to this object. The canvas title is set to the event name, and the event information appended to the output log.

readNext() Calls AEventSource.readNext();

readPrevious() Calls AEventSource.readPrevious();

5.2 AEventSource

Interface, 2 implementing classes:

- AXMLFileEventSource
- AXMLZipEventSource

5.3 AXMLFileEventSource

readEvent(String eventSource) Checks the mode and calls the read(FileInputStream eventSource, String eventName) method from an instance of AUserEventFromXML or AEventFromXML accordingly.

readNext() This checks the fileName for an event number (file_no.xml), increments this, and replaces the leading zeros as necessary. The same methods as in readEvent are then called.

readPrevious() The same as readNext except decrement the event number.

5.4 AXMLZipEventSource

readEvent(String eventSource) Uses the java.util.zip packages to look for XML files within zipfiles. It counts the number of XML files and creates a 1D array of this size. It then loop through the zip file again, and fills the array with the ZipEntry objects. Then for the first object it calls the public AEvent read(FileInputStream eventSource, String eventName) method from an instance of AEventFromXML.

readNext() Increments the eventIndex by one and calls the read(FileInputStream eventSource, String eventName) method on the corresponding ZipEntry object in the array.

readPrevious() Decrements the eventIndex by one. Same method call as above.

5.5 AEventFromXML

This is a Singleton Class that extends the DefaultHandler from the SAX API.

AEventFromXML() This private constructor is called when the getInstance method from the class is called. It creates a SAXParserFactory, sets it to validating and namespaceaware, then creates and instance of the SAXParser from this. The error handler (AXMLErrorHandler) is set here.

getInstance() This creates a new instance of the class, or if this is present, returns the current instance.

read() This method takes an InputStream and String (eventName) as arguments. It creates a new AEvent object, and calls the parse() method from the SAXParser. AEvent.finalizeEvent is then called if no errors were thrown by the parser.

The rest of this class is the implementation for the default handler.

When reading the event from the XML file, the XML parser knows it is reading an element(tag) e.g. <S3D><rho> but does not know the structure of the document, i.e. the role of that tag within the document, this information is what is kept in an Attribute of the class called state. Also when reading an array of numbers, the XML parser gives it to the program as multiple arrays of characters so the program must also keep state information to understand when it has finished reading an array. The XML parsers are designed as finite state machines.

An int is used to represent the state (WAITING_EVENT(0), WAITING_DATA(1), WAITING_ARRAY(2)), and a boolean is used to record if an error has been encountered

startDocument This methods is called when the start of document is encountered. This checks if error=false, then sets the state to WAITING_EVENT.

endDocument This does nothing more than checks if error=false and writes to the log.

startElement This method is called at the start of each tag encountered. It is given the URI, localname of the tag, qualified name (includes the namespace)of the tag, and any attributes of the tag by the parser.

This has different functions depending on the current value of the state attribute. An explanation of each of these is given below:

Example layout of the XML file:

line 1 <?xml version="1.0"?>
line 2 <?ATLAS Release: "6.0.3"?>
line 3 <!DOCTYPE Event SYSTEM "event.dtd">

line 4 <Event version= "6.0.3">
line 5 <TRT Count = "4557">
line 6 <barcode>....</barcode>
line 7 <driftR>....</driftR>
line 8 <id>.....</id>
line 9 <kine>....</kine>

line 10 <phi>....</phi>
.....
</TRT>

<?xml version="1.0"?> When the XML parser encounters this line then the startDocument method is called. Here the state kept by the AEventFromXML object is set to WAITING_EVENT.

<!DOCTYPE Event SYSTEM "event.dtd"> Here resolveEntity() is called to set the DTD to that in event.dtd.

<Event version= "6.0.3"> startElement() is called, and the state changed to WAITING_DATA.

<TRT Count = "4557"> startElement() is called again, and the int arraylength attribute of the class set to the value given (4557). The class keeps a AHashMap object called parameters. The count value is placed into this AHashMap (parameters.put("numData", new Integer(arrayLength))). The state is now set to WAITING_ARRAY.

<barcode> At line 6, startElement is called again, this time the type is requested (must be gained from the .dtd file) and an AArrayParser of this type is created and set to the currentArrayParser. There are three types AStringArrayParser, AFloatArrayParser and AIntArrayParser. The size of the array is set from the count obtained in line 5, and an attribute called multiple. Suppose there are 25 tracks then arrayLength=25. For most attributes of the track (e.g. d0) multiple = 1 but for the covariance matrix multiple is 15 so the total size of that array is 25*15. A check is made that multiple does not equal NaN (not a number).

For each chunk of data the parser reads from the file the characters() method is called. This passes the data to the currentArrayParser. When the </barcode> tag is reached the endElement method is called.

endElement When </barcode> is encountered the method parameters.put(currentArrayName, arrayParser.getArray()); is called. This puts this array of data into the current parameters AHashMap.

If </TRT> is encountered then the AEvent.add(); method is called and a new object of the correct type (ATRTData for example) is created with the parameters array as the argument. The state is then changed back to WAITING_DATA.

characters This method takes action for each chuck of character data, as the data between the startElement and endElement flags is split up by the XML parser as it chooses. This method just passes the character array it is given to the current array parser, using its parse method.

resolveEntity This method, when it encounters the line containing the name of the dtd file, returns this an as inputSource object.

checkPhiRange Method to check the phi values are in the range 0 - 2Pi.

5.6 AEvent

Attributes: private Map data, private String name

Methods:

Constructors The Constructor for this class sets the name, creates a new HashMap and adds to this a ARVxData object (Reconstructed Vertex Data). This is the only data added at this point, as it is something that can be created dynamically by the program. A Constructor that sets the name to an empty string is also avaliable.

public void add(Object dataSet) This method adds the given object to its Map of data.

public AData get(String name) will return any data type requested, or individual methods for each are also implemented (e.g. getTRTData()).

public AData{} getData() This method will return all the different types of data in the event in an array.

public List getCalorimeters(AProjection projection) This method returns a List of ACalorimeterData that are visible for a given projection. If the status of the calorimeter data is visible then the calorimeter data is added to the returned List. In the case of the XY projection and the FR projection with a mode above or equal to 9, then these are the views of the Forward Calorimeter. Here mode 9 is the EM FCAL, mode 10 is the 1st Hadronic layer and 11 is the 2nd Hadronic layer. This is the only special case here.

public List getCalorimeters() This simply checks the status of each calorimeter data and adds it to the returned List.

public List getHitsAndTracks(AProjection projection)

public List getElectromagneticCalorimeters() Returns ALArData, AFCALData, AFastCalData.

public List getHadronicCalorimeters() Returns ATILEData, AHECData, AFCALData and AFastCalData.

public AEvent finalizeEvent() This sets the currentEvent in the AEventManager to this AEvent, it then calculates the primary vertex position using getSTrData().calculatePrimaryVertex(). The finalizeConstruction method is then called in turn on each AData object, and lastly the getS3DData().reallyFinalizeConstruction method is called. The AListManager is reset and the AEvent returned.

public String{}{} getInfo() *still to be understood*

public double{} getPrimaryVertex() Returns the [x,y,z] positions of the primary vertex.

6 Data

6.1 AData