# XML Querying and Communication Classes for the VizIR Framework

Geert Fiedler (fiedler@ims.tuwien.ac.at)

2004-03-11

## *Abstract*

This document gives an introduction into the usage of the XML querying classes (based on the Multimedia Retrieval Markup Language (MRML) [1]) and communication classes of the VizIR Framework (based on Java web services). It describes building and validating XML representations of the query data as well as communication between querying client and server.

## *Table of Contents*

## *Multimedia Retrieval Markup Language*

Communication of user interfaces and query engines in VizIR is based on loose coupling of components through using the Multimedia Retrieval Markup Language (MRML). MRML is an XML-based language developed by the University of Geneva (see [1] for more information). We extended MRML with additional elements to become able using it with the querying paradigms implemented in the VizIR framework.

The following code example illustrates a simple query formulation (from [2]):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mrml xsi:noNamespaceSchemaLocation="http://cbvr.ims.tuwien.ac.at/ vizir_elements_new.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <begin-transaction transaction-id="1"/>
    <logicalQuery>
        <clusterDefinition>
            <clusterRestriction>
                <clusterDimension lowerBound="1.1" upperBound="1.5">
                    <mediaGroup id="1">
                            <mediaObject iconLocation=" http://cbvr.ims.tuwien.ac.at/icons/ img1.gif"
                            dataLocation="http://cbvr.ims.tuwien.ac.at/icons/ thumb1.gif"/>
                            <descriptor value="1.0" name="ColorHistogram" distanceValue="10.0"/>
                    </mediaGroup>
                    <descriptor value="0.5" name="ColorHistogram" distanceValue="5.0"/>
                </clusterDimension>
            </clusterRestriction>
        </clusterDefinition>
    </logicalQuery>
    <end-transaction transaction-id="1"/>
</mrml>
```

This construct defines a query (on the *collection* defined elsewhere in the MRML script) with a single feature. A colour histogram is used to find all media objects that have a distance to the query example *'img1.gif'* (represented by the icon *'thumb1.gif'*) that is smaller than *'0.5'*. If we would like to retrieve all objects that fulfil this condition *and* a second one, we would put the second *clusterRestriction* in the same *clusterDefinition*. If we would like to retrieve all objects that fulfil the first *or* the second criteria, we would put the second one in a new *clusterDefinition*. See [2] for more information on our MRML extension for Visual Information Mining and the appendix for the extended MRML document type definition.

## *Working with the MRML classes*

As mentioned above MRML defines an XML format. In order to use XML with Java it is necessary to represent XML tags and attributes by classes and resources. Based on such classes it is possible to map the original XML structure to the according Java representation (parsing) and vice versa (writing). Additionally, the classes provide methods to access the attributes. The following two subsections describe parsing and writing. The VizIR source tree provides classes for all MRML elements but,

alternatively, it is possible to create a new class set from the DTD. Refer to the appendix for information on derivation of the MRML class tree.

## Parsing MRML

The following libraries have to be available for using the VizIR MRML classes (available from [3]):

- .\jwsdp-1.3\common\lib
- .\jwsdp-1.3\jaxb\lib
- .\jwsdp-1.3\jwsdp-shared\lib

Below, based on the above example we explain how to create and link the MRML objects. The first method to create an object tree is building and parsing a string containing the MRML structure. Take care that the DTD is available, because the parser validates the string.

```
String mrmlString = new String(
   "<!DOCTYPE mrml SYSTEM \"file://vizir.dtd\">" +
   "<mrml>" +
   "<begin-transaction transaction-id=\"1\"/>" +
   "<logicalQuery><clusterDefinition>" +
   "<clusterRestriction><clusterDimension " +
   "lowerBound=\"1.1\"   upperBound=\"1.5\">" +
   "<mediaGroup id=\"1\"><mediaObject dataLocation=\"MYURL\"" +
   " iconLocation=\"testURLIconLoc\">" +
   "<descriptor distanceValue=\"10.0\" name=\"Test\" value=\"1.0\"/>" +
   "</mediaObject></mediaGroup>" +
   "<descriptor distanceValue=\"1.0\" name=\"TestMediaGroup\"" +
   " value=\"1.0\"/>" +
   "</clusterDimension></clusterRestriction>" +
   "</clusterDefinition></logicalQuery>" +
   "<end-transaction transaction-id=\"1\"/>" +
   "</mrml>"
);
```

Calling the method `mrmlStringParser` creates a vector that contains the MRML objects. It is necessary to call the method `makeObjectTree` in order to rebuild the original XML structure and get the root element.

```
try {
   messageVector = mrmlReader.mrmlStringParser(mrmlString);
   MrmlData data = new MrmlData(messageVector);
   Mrml mrmlData = data.makeObjectTree();
} catch ( Exception e ) {
   System.out.println(e);
}
```

## Writing MRML

Alternatively, it is possible to build the MRML object tree by using the MRML classes and their linking methods. The following example shows how to build a MRML object tree from scratch.

```
import org.vizir.mrml.*;
import org.vizir.mrml.xml.mrml.*;
```

First, variables are defined for the MRML representation of the query.

```
Mrml root = null;
BeginTransaction beginTransactionElement = null;
EndTransaction endTransactionElement = null;
LogicalQuery logicalQueryElement = null;
ClusterDefinition clusterDefinitionElement = null;
ClusterRestriction clusterRestrictionElement = null;
ClusterDimension clusterDimensionElement = null;
MediaGroup mediaGroupElement = null;
MediaObject mediaObjectElement = null;
Descriptor descriptorElement = null;
```

`ObjectFactory` is used to instantiate new objects. The class `MakeMrmlElements` provides some convenience methods for object creation.

```
// the factory
ObjectFactory object = new ObjectFactory();
MakeMrmlElements makeMrmlElements = new MakeMrmlElements(object);
```

The first step is creation of a root object. This root object is similar with a root object of a linked list. All other elements are managed similarly to collection classes.

```
try {
   // mrml
   root = object.createMrml();

   // BeginTransaction
   beginTransactionElement =
      makeMrmlElements.makeBeginTransactionElement("1");
```

If a MRML element has only one child element, get- and set-methods can be used for manipulation:

```
   root.setBeginTransaction(beginTransactionElement);

   endTransactionElement =
      makeMrmlElements.makeEndTransactionElement("1");
   root.setEndTransaction(endTransactionElement);

   // LogicalQuery
   LogicalQuery LogicalQueryElement = object.createLogicalQuery();
   root.setLogicalQuery(LogicalQueryElement);

   // ClusterDefinition
   clusterDefinitionElement = object.createClusterDefinition();
```

If MRML elements have two or more child elements all objects are added to a list. So it is possible to manage the objects with methods from the collection classes.

```
   // add ClusterDefinition to LogicalQuery
   LogicalQueryElement.getClusterDefinition().
      add(clusterDefinitionElement);
   // ClusterRestriction
   clusterRestrictionElement = object.createClusterRestriction();
   // add ClusterRestriction to ClusterDefinition
   clusterDefinitionElement.getClusterRestriction().
      add(clusterRestrictionElement);
   // first ClusterDimension
   clusterDimensionElement =
      makeMrmlElements.makeClusterDimensionElement(1.1,1.5);
   // add ClusterDimension to ClusterRestriction
   clusterRestrictionElement.getClusterDimension().
      add(clusterDimensionElement);

   //Descriptor
   descriptorElement = makeMrmlElements.
      makeDescriptorElement("Color Histogramm",5.0,0.5);
   // set the descriptor for the clusterDimensionElement
   clusterDimensionElement.setDescriptor(descriptorElement);

   // MediaGroup
   mediaGroupElement = makeMrmlElements.makeMediaGroupElement("1");
   // MediaObject
   mediaObjectElement = makeMrmlElements.makeMediaObjectElement(
      "http://cbvr.ims.tuwien.ac.at/icons/img1.gif",
      "http://cbvr.ims.tuwien.ac.at/icons/thumb1.gif"
   );

   // Descriptor for the MediaObject
   descriptorElement = makeMrmlElements.makeDescriptorElement(
      "Color Histogramm",10.0,1.0
   );

   // add Descriptor to the MediaObject
   mediaObjectElement.getDescriptor().add(descriptorElement);
   // add MediaObject to the MediaGroup
   mediaGroupElement.getMediaObject().add(mediaObjectElement);
   // add MediaGroup to the ClusterDimension
   clusterDimensionElement.setMediaGroup(mediaGroupElement);
} catch ( JAXBException e ) {
   System.out.println("Error in creating mrml elements: " + e);
}

MrmlWriter writer = new MrmlWriter();
String ContentOfXmlString = writer.createMrmlString(root,
   "http://cbvr.ims.tuwien.ac.at/vizir_elements_new.xsd"
);

//Print out the MRML data for validation
System.out.println(ContentOfXmlString);
```

## *VizIR XML Communication*

In addition to the libraries named above, ".\jwsdp-1.3\saaj\lib" is needed to use the VizIR communication classes (can be downloaded from [3]). The main communication classes are `XMLReceive` and `XMLSend`. `XMLReceive` runs in a thread and implements all methods for receiving data. Messages can be received as strings as MRML root elements. Whenever a reception process is finished an event is fired to signal that data is available. For this reason the receiver needs to register an event handler. If the received data contains a query definition the event returns a vector that stores the MRML objects. To create valid MRML data from this vector it is necessary to call the method `makeObjectTree` (as described above). If the received data is a string object the vector is set to *null* and the event returns a string object. `XMLSend` implements methods for sending the data. When data is sent the receiver sends an acknowledge to indicate successful reception. The following code segments show how receiver and sender are used.

### Receiver

Receiver classes need to implement an event listener for data reception.

```
public class SimpleReceiver implements XMLEventListener {
   public int PortNumber = 8001;
```

The message vector needs to be defined as a global variable. If an event is fired the MRML objects are returned in this vector.

```
   public Vector messageVector = new Vector();
```

The event returns either a vector with serialized MRML objects or a string object. The event handler method is called automatically when an event is fired.

```
   public void handleXMLMessagingEvent(XMLMessagingEvent event) {
      // get the size of the vector
      messageVector = event.getXMLVector();

      if (messageVector==null) {   // string object, vector is empty
         message = event.getMessage();
         //...
      } else { // MRML root object
         try {
            MrmlData data = new MrmlData(messageVector);
            Mrml mrmlData = data.makeObjectTree();
         } catch ( Exception e ) {
            System.out.println(e);
         }
      }
   }

   synchronized public void ReceiveMessageTest {
      // new socket
      ServerSocket serverSocket = new ServerSocket(PortNumber);
      Socket socket = null;
      XMLReceive receiver = null;
```

```
      try {
         // listen for data
         socket = serverSocket.accept();
         receiver = new XMLReceive(socket,this);
         receiver.start();
         wait(1000);
         receiver.stopReceiver();
      } catch (Exception e) {
         //...
      }
      //...
   }
}
```

## Sender

```
public class SimpleSender {
   public int PortNumber = 8001;
   public String ServerName = new String("localhost");
   public Mrml mrmlData = null;

   // create some mrml data...

   XMLSender sender = new XMLSender(ServerName,PortNumber);
   sender.setRespondTimeOut(1000);
   // optional it is possible to send a string object
   // sender.sendString("A STRING TO SEND");
   sender.sendXML(mrmlData,"d://mrml_dtd_xml//vizir_elements_new.xsd");
}
```

# Appendix

## *How to create the MRML classes*

### JAXB

JAXB (Java API for XML Binding) makes XML easy to use in Java by compiling an XML schema into one or more classes. The package (currently, version 1.3) can be downloaded from http://www.java.sun.com. For further information on the binding process refer to http://java.sun.com/webservices/docs/1.3/jaxb/xjc.html.

### Instructions for Windows operating systems:

1. Setting the system variables

set JAVA_HOME=<*your J2SE SDK installation directory*>
set JWSDP_HOME=<*your JWSDP1.2 installation directory*>
set JAXB_HOME=%JWSDP_HOME%\jaxb
set PATH=%JAXB_HOME%\bin;%JWSDP_HOME%\jwsdp-shared\bin;%PATH%

2. Execution

%JAXB_HOME%\bin\xjc.bat -p org.vizir.mrml.xml.mrml <xmlSchema.xsd>
(do not forget to overwrite the old MRML classes with the new ones!)

## *Extended MRML document type definition*

This appendix contains the MRML extension used in the VizIR framework. It is based on the MRML definition in [1]. Below, the most relevant extensions to the original MRML DTD are explained.

```
<!ELEMENT mrml (begin-transaction?, (get-configuration | configuration-description | get-sessions |
session-list | open-session | rename-session | close-session | delete-session | get-collections |
collection-list | get-algorithms | algorithm-list | get-property-sheet | property-sheet | configure-session |
query-step | query-result | user-data | error | mediaGroup | logicalQuery)?, end-transaction?)>
<!ATTLIST mrml
    session-id CDATA #IMPLIED
    transaction-id CDATA #IMPLIED>
<!ELEMENT algorithm-list (algorithm*)>
<!ELEMENT algorithm (algorithm*, property-sheet?, query-paradigm-list?, allows-children?)>
<!ATTLIST algorithm
    algorithm-id CDATA #REQUIRED
    collection-id CDATA #IMPLIED
    algorithm-name CDATA #REQUIRED
    algorithm-type CDATA #IMPLIED>
<!- ... ->
```

The MRML extensions are simply added as subtags of the mrml tag. VizIR visual mining queries are defined using these tags (see [2]).

```
<!ELEMENT logicalQuery (clusterDefinition+)>
```

`clusterDefinition` defines the cluster of related objects that fulfil the condition stated in `clusterDimension`. If you would like to retrieve all objects that fulfil this condition AND others, the `clusterRestrictions` have to be put in the same `clusterDefinition`. If you want to retrieve all that fulfil the condition OR another, the `clusterRestrictions` has to be put in another `clusterDefinition`.

```
<!ELEMENT clusterDefinition (clusterRestriction+)>
<!ELEMENT clusterRestriction (clusterDimension+)>
<!ELEMENT clusterDimension (mediaGroup, descriptor)>
<!ATTLIST clusterDimension
    lowerBound CDATA #REQUIRED
    upperBound CDATA #REQUIRED>
```

Media objects can be defined as *user-relevance-elements* or as *query-result-elements*.

```
<!ELEMENT mediaObject (descriptor*)>
<!ATTLIST mediaObject
    dataLocation CDATA #REQUIRED
    iconLocation CDATA #REQUIRED>
```

The following construct is used for the definition of media groups.

```
<!ELEMENT mediaGroup (mediaObject+)>
<!ATTLIST mediaGroup
    id CDATA #REQUIRED
    type CDATA #IMPLIED>
```

For the definition of features `descriptor` is used. `DistanceValue` is a special field that is used when media objects are grouped to describe the layout in distance space.

```
<!ELEMENT descriptor EMPTY>
<!ATTLIST descriptor
    name CDATA #REQUIRED
    value CDATA #IMPLIED
    distanceValue CDATA #IMPLIED>
<!- ... ->
```

# References

[1] Multimedia Retrieval Markup Language Website, http://www.mrml.net, last visited 2004-03-10

[2] Eidenberger, H., and Breiteneder, C., A Framework for User Interface Design in Visual Information Retrieval, IEEE International Symposium on Multimedia Software Engineering, Newport Beach, USA, 2002.

[3] Java Web Development Kit, http://java.sun.com/webservices/downloads/webservicespack.html, last visited 2004-03-10