8.3 The ProcessorModel

The simplest way to create a Processor is by using a ProcessorModel. A ProcessorModel abstracts the essential information that is needed by the Manager factory class to construct a Processor. The class diagram of the ProcessorModel is shown in figure 8.7.

javax.media.ProcessorModel
<<constructor>> public ProcessorModel() public ProcessorModel(Format [] formats, ContentDescriptor outputContentDescriptor) public ProcessorModel(DataSource inputDataSource, Format[] formats,ContentDescriptor outputContentDescriptor) public ProcessorModel(MediaLocator inputLocator, Format[] formats,ContentDescriptor outputContentDescriptor) << methods>> public int getTrackCount(int availableTrackCount) public Format getOutputTrackFormat(int tIndex) public boolean isFormatAcceptable(int tIndex, Format tFormat) public ContentDescriptor getContentDescriptor() public DataSource getInputDataSource() public MediaLocator getInputLocator()

Figure : 8.6 The class definition of ProcessorModel.

The class ProcessorModel is defined as,

public class ProcessorModel extends java.lang.Object.

Apart from a null constructor the ProcessorModel has three other constructors.The constructor,

public ProcessorModel(DataSource inputDataSource,
Format[] formats, ContentDescriptor outputContentDescriptor)

can be used to construct a Processor that can take media from the inputDataSource, set the track format of its tracks as given by the parameter formats and can give an output defined by the outputContentDescriptor. Another constructor of the ProcessorModel is identical to the above excepting that the media source is specified by a MediaLocator rather than by a DataSource. The third constructor of the ProcessorModel is also identical to the above-mentioned constructor but with no media source being mentioned.

To create a realized processor by using the ProcessorModel we have to invoke the Manager's static method,

public static Processor createRealizedProcessor(ProcessorModel model)
throws java.io.IOException, NoProcessorException,
CannotRealizeException.

The Manager will attempt to create a Realized Processor as specified by the given ProcessorModel. The method throws a NoProcessorException if either (I) the ProcessorModel is an invalid model or (ii) suitable PlugIns are not available to construct a Processor satisfying the specified model. Suppose the ProcessorModel has no specified DataSource then the createRealizedProcessor method using such as ProcessorModel will assume a suitable capture device as the DataSource of the Processor.

Example 8.2 given here illustrates construction of a Realized Processor using the ProcessorModel.

```
/* Example 8.2. Program illustrates the use of the ProcessorModel to construct a Processor.
*/

import javax.swing.*;
import java.awt.*;
import javax.media.*;
import javax.media.format.*;
import java.net.*;
import javax.media.protocol.*;
import java.awt.event.*;
import java.io.*;

public class procModel {

MediaLocator ml;
static DataSource ds;
static Processor  proc= null;

public procModel(String locatorString){

        JFrame frame = new JFrame( "Processor Model");
         if( locatorString != null){
                  ml = new MediaLocator( locatorString);
                 try{
                          ds = Manager.createDataSource( ml);
                 }catch( IOException ioe){ System.out.println(ioe);
                 }catch(NoDataSourceException npe){ System.out.println (npe);
                 }
           } else {
                 ds = null;
        }
        Format mfs[] = new Format[1];

        //mfs[0] = new VideoFormat( VideoFormat.YUV );
        //mfs[0] = new VideoFormat( VideoFormat.RGB );
        //mfs[0] = new VideoFormat( VideoFormat.H263 );

        mfs[0] = new AudioFormat( "LINEAR", 8000, 16, 2 );
        //mfs[0] = new AudioFormat( "LINEAR", 11025, 16, 1 );
        // mfs[0] = new AudioFormat( AudioFormat.GSM);
        //mfs[0] = new AudioFormat( AudioFormat.ULAW);
```

```
        //Format mfs[] = new Format[2];
        //mfs[0] = new VideoFormat( VideoFormat.RGB );
        //mfs[1] =  new AudioFormat( "LINEAR", 8000, 16, 2 );

        try{
                ProcessorModel model  =  new ProcessorModel( ds,mfs, null );
                proc =Manager.createRealizedProcessor( model) ;
        } catch( Exception e) {
                System.out.println(" Processor not realized" + e );
        }

        Component control = proc.getControlPanelComponent();
        if( control != null){
                frame.getContentPane().add( control, BorderLayout.SOUTH);
        }

        Component visual = proc.getVisualComponent();
         if( visual != null) {
                frame.getContentPane().add( visual, BorderLayout.CENTER);
        }

        frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent we) {
                        proc.stop();
                        proc.close();
                        System.exit(0);
                }
        });
        frame.pack();
        frame.setVisible(true);
        proc.start();
}

public static void main( String args[] ) {
        if( args.length == 1 ) {
                procModel pm = new procModel( args[0]);
        } else if( args.length == 0){
                procModel pm = new procModel(null);
        } else{
           System.out.println("Usage : java  procModel  [ locatorString] " );
        }
}

}
```

Following is the explanation for example 8.2. The main objective of the program is to construct the Processor using a ProcessorModel. The ProcessorModel should encapsulate three important specifications answering the following questions related to the construction of the Processor. (I) Which is the source of the media to the Processor? (ii) How do we want to process the media ?, and (iii) Where do we want to output the processed media that comes out

of the Processor ? The program specifies these three specifications as the following parameters to the constructor of the ProcessorModel.

(i) Media source

The constructor of the ProcessorModel takes a MediaLocator or a DataSource or a null as the source of the media. Example 8.2 optionally accepts the locatorString of the media source for which a Processor will be created. If the user specifies the locatorString then a MediaLocator object and subsequently a DataSource is created. This DataSource is used in the specification of the ProcessorModel.

In case the user does not specify the locatorString then we pass a null value for the parameter DataSource used in the specification of the ProcessorModel. Note that if no DataSource is specified or if the DataSource is specified as null in the constructor of the ProcessorModel then the Processor created using such a ProcessorModel will use suitable capture source(s).

(ii) Desired Processing

We can specify the desired processing by specifying the desired format(s) of the track(s) of the Processor. This is done by specifying in the ProcessorModel an array of Format object(s) representing the format(s) to be set on the track(s) of the Processor.

If we know that the media source has only an audio or a video track then correspondingly we can specify the desired audio or video format of the track. In that case the array of Formats to be specified in the ProcessorModel should have only one element.

If the source has both the audio and video tracks we can specify an array of two Format objects one being of AudioFormat type and another being of the VideoFormat type. (iii) Where do we want render the Processor's output ?

One may like to render the Processor's output into a file or as a network stream or on an output device. In this example 8.2 let us render the media on an output device. In other words we would like to use the Processor as a Player. A Processor will behave as a Player if its output ContentDescriptor is set as null (refer the section 8.4.6 ContentDescriptor for more details). Therefore we set the desired output ContentDescriptor in the ProcessorModel as null.

The Manager attempts to construct a realized Processor adhering to the ProcessorModel described by the above three specifications. Note that the Manger may fail to create a realized Processor for three different reasons and accordingly three possible exceptions can be thrown and we need to handle them.

If the Processor is successfully created it will also be realized and therefore we can get the visual and control components from it and add it to our application window. The Processor is closed when the application frame is closed.

Compile example 8.2 and run it. First you can experiment by running example 8.2 with no command line arguments. Then a Processor will be created for a suitable capture device. You may try the following other possibilities,

(i) Define mfs as an array of one element and specify that element as one of the AudioFormats or VideoFormats.

(ii) Define mfs as an array of two elements and specify two elements such that one is of the type AudioFormat and another is of the type videoFormat.

You can also run example 8.2 by passing a locatorString as a command line argument. Try using the locatorString of an audio file (example a Wav file). For processing and playing an audio file define mfs as an array with one element and specify that element as of the type AudioFormat. Similarly you can try using a video only file (You can create a video only file using JMStudio). Finally you can specify the locatorString of a file having both audio and video (example mpg file) and proceed in as the case (ii) above.

The ProcessorModel is a simple method for specifying the desired processing of a Processor. However it does not offer you the full control over the processing performed by the Processor. To be specific

(i) It does not allow you to add Effect PlugIns.

(ii) Supposing two or more similar codecs are registered in the JMF registry. They are similar in the sense that both of them does identical format conversion. However let us suppose that one codec has better peformance over the rest. In this case the ProcessorModel will use any one of the two or more candidate codecs. You have no control over which one to be used.

(iii) At last the createRealizedProcessor () is a blocking call.

For all these reasons it is frequently needed to step through the various stages in the construction of a Processor. Then we can have a full control over its processing.