Java Media Framework
Capture and Media Conversion

Dave Price
Computer Science
University of Wales, Aberystwyth

Interacting with the CaptureDeviceManager

- **public static java.util.Vector getDeviceList(Format format)**
  - Gets a list of CaptureDeviceInfo objects that correspond to devices that can capture data in the specified Format
- **public static CaptureDeviceInfo getDevice(java.lang.String deviceName)**
  - Gets a CaptureDeviceInfo object that corresponds to the specified device.
- plus various others to add and delete devices

CaptureDeviceInfo

- **public java.lang.String getName()**
  - Gets the name of this device. The name might include the device name, vendor name, and a version number.
- **public MediaLocator getLocator()**
  - Gets the MediaLocator needed to create a DataSource for this device through the Manager. The MediaLocator is unique--no two devices can use the same locator.

CaptureDeviceInfo continued

- **public Format[] getFormats()**
  - Gets a list of the output formats supported by this device.
  - Returns: A Format array that contains the output formats supported by this device.

Format

- Direct Known Subclasses:
  - AudioFormat, ContentDescriptor, VideoFormat
- **public java.lang.String getEncoding()**
  - Gets the uniquely-qualified encoding name for this Format. In the reference implementation of JMF, these strings follow the QuickTime codec strings.

Formats - continued

- **public java.lang.String toString()**
  - Gets a String representation of the Format attributes. For example: "PCM, 44.1 KHz, Stereo, Signed".
- **public boolean matches(Format format)**
  - Checks whether or not the specified Format matches this Format. Matches only compares the attributes that are defined in the specified Format, unspecified attributes are ignored.
- plus various other methods

Interacting with the PlugInManager

- **public static java.util.Vector getPlugInList(Format input, Format output, int type)**
  - Builds a list of plug-ins that satisfy the specified plug-in type and input and output formats. Either or both of the formats can be null. If input is null, getPlugInList returns a list of plug-ins of the specified type that match the output format. If output is null, getPlugInList returns a list of plug-ins of the specified type that match the input format. If both parameters are null, getPlugInList returns a list of all of the plug-ins of the specified type.

Values for int type field

- **PlugInManager.DEMULTIPLEXER**
- **PlugInManager.CODEC**
- **PlugInManager.EFFECT**
- **PlugInManager.RENDERER**
- **PlugInManager.MULTIPLEXER**

Interacting with Manager

- **public final class Manager extends java.lang.Object**
  - Manager is the access point for obtaining system dependent resources such asPlayers, DataSources, Processors, DataSinks, the system TimeBase, thecloneable and merging utility DataSources.

Capturing via a MediaLocator and just playing to the screen

- **public static Player createPlayer(MediaLocator sourceLocator) throws java.io.IOException, NoPlayerException**
  - Create a Player for the specified media.
- Look at the code for my example SimpleCapture.java, it finds a device by name and then uses the above method.

Finding a Capture Device that supports a certain VideoFormat

- **public VideoFormat(java.lang.String encoding, java.awt.Dimension size, int maxDataLength, java.lang.Class dataType, float frameRate)**
  - Constructs a VideoFormat with the specified attributes.
  - Parameters:
  - encoding - A String that describes the encoding type for thisVideoFormat.
  - size - The size of a video frame.
  - maxDataLength - The maximum length of a data chunk.
  - dataType - The type of data. For example, byte array.
  - frameRate - The frame rate.

My use of VideoFormat in RGBCapture.java

- VideoFormat rgbFormat = new VideoFormat(VideoFormat.RGB, new Dimension(384,288), Format.NOT_SPECIFIED, Format.byteArray, Format.NOT_SPECIFIED);

Using Manager to create a DataSource

- **public static DataSource createDataSource(MediaLocator sourceLocator) throws java.io.IOException, NoDataSourceException**
- Create a DataSource for the specified media.

Getting FormatControl objects

- **public interface CaptureDevice**
  - CaptureDevice is the interface for all capture devices.
- **public FormatControl[] getFormatControls()**
  - Returns an array of FormatControl objects. Each of them can be used to set and get the format of each capture stream. This method can be used before connect to set and get the capture formats.

FormatControl

- **public interface FormatControl extends Control**
  - The FormatControl interfaces is implemented by objects which supports format setting.
- **public Format setFormat(Format format)**
  - Sets the data format.
  - Returns:
  - null if the format is not supported. Otherwise returns a format that most closely matches the specified one.

Using Manager to create a player from a DataSource

- **public static Player createPlayer(DataSource source) throws java.io.IOException, NoPlayerException**
  - Create a Player for the DataSource.

Creating a player from a ProcessorModel

- **public static Processor createRealizedProcessor(ProcessorModel model) throws java.io.IOException, NoProcessorException, CannotRealizeException**
  - Create a Realized Processor for the specified ProcessorModel.
  - This method accepts a ProcessorModel that describes the required input and/or output format of the media data. It is a blocking method and returns only after the Processor reaches the Realized state.

What's a ProcessorModel

- **public class ProcessorModel extends java.lang.Object**
  - Encapsulates the basic information required to create a Processor. A ProcessorModel can be passed to Manager.createRealizedProcessor to construct a Processor.

Constructing a ProcessorModel

- **public ProcessorModel(Format[] formats, ContentDescriptor outputContentDescriptor)**
  - Creates a ProcessorModel for the specified track formats and output content-type. This constructor creates a ProcessorModel that can be used to construct a Processor for capturing media data.
  - Parameters:
  - formats - An array of Format objects that contains the desired track formats.
  - outputContentDescriptor - A ContentDescriptor that describes the desired output content-type.

My use of ProcessorModel in ModelCapture.java

```
Format [] formats = new Format[2];
formats[0] = new
  AudioFormat(AudioFormat.IMA4);
formats[1] = new
  VideoFormat(VideoFormat.JPEG);
FileTypeDescriptor outputType = new
  FileTypeDescriptor(
  FileTypeDescriptor.QUICKTIME);
ProcessorModel processorModel =
      new ProcessorModel(formats,outputType);
```

Acquiring the DataSource output from a Processor

- **public DataSource getDataOutput() throws NotRealizedError**
  - Gets the output DataSource from the Processor. The output DataSource is the output connection through which the processed streams are supplied. The output DataSource returned by the Processor is in the Connected state.

Saving to file - creating a DataSink

- **public static DataSink createDataSink(DataSource datasource, MediaLocator destLocator) throws NoDataSinkException**
  - Create a DataSink for the specified input Datasource and destination Medialocator.

Making a DataSink

```
DataSource outputData = processor.getDataOutput();
 MediaLocator outMediaLocator = new
   MediaLocator(outputFile);
 try {
    outSink = Manager.createDataSink( outputData,
                                        outMediaLocator);
     outSink.open();
 } catch (Exception e) {
     System.out.println("Exception occured " + e);
     System.exit(0);
 }
 outSink.addDataSinkListener(this);
```

Starting capture etc

```
System.out.println("About to call start on processor");
addVisualElements();
setVisible(true);
try {
   outSink.start();
   processor.start();
} catch (Exception e) {
   System.out.println("Exception on start " + e);
}
System.out.println("Called start on processor");
waitForState(processor.Started);
System.out.println("processor started");
```

DataSinkListener

- **public interface DataSinkListener**
  - DataSinkListener is an interface for handling asynchronous events generated by DataSink.
- **public void dataSinkUpdate(DataSinkEvent event)**
  - This method is called when an event is generated by a DataSink that this listener is registered with.
  - Parameters:
  - event - The event generated.

My example method

```
public void dataSinkUpdate(DataSinkEvent dse) {
    if (dse instanceof EndOfStreamEvent) {
        System.out.print("EndOfStream event");
        outSink.close();
    } else {
        System.out.print("datasink event was " +
                                                dse);
    }
}
```

Getting Processor TrackControls

- **public TrackControl[] getTrackControls() throws NotConfiguredError**
  - Gets a TrackControl for each track in the media stream. This method can only be called once the Processor has been configured.
  - Returns:
  - An array of TrackControl objects. An empty array is returned if there is no TrackControl available for this Processor.

Getting a Tracks support formats

- **public Format[] getSupportedFormats()**
  - Lists the possible input formats supported by this plug-in.
  - Returns:
  - an array of the supported formats