제 3 강좌 : JMF의 새로운 기능 - Live 데이터 획득 및 처리

지난 달 강좌를 통하여 JMF에서의 자세한 이벤트 처리 방식을 살펴보았으며, JMF에서 기본적으로 제공되는 User Interface의 재설계를 통한 Custom User Interface의 구현방법과 내부 동작원리, 사용자의 UI 컨트롤에 대한 재적용 방법에 대하여 알아보았다. 또한 Swing과 결합된 멀티미디어 데이터재생기를 설계함으로서 Swing을 이용한 JMF의 외형꾸미기를 구현하였다. 이번 강좌에서는 JMF 2.x 버전에서 새롭게 추가된 내용인 멀티미디어 데이터의 캡쳐 작업 즉, 오디오 데이터와 비디오 데이터에 대한 라이브 데이터의 획득을 통하여 프로그래밍과 이에 관한 세부적인 내용을 설명하고 이를 구현한 프로그래밍을 해보고자 한다. 강좌에 관한 문의나 관련 프로그램은 아래의 홈페이지를 참조하기 바란다.

이재훈 전임연구원
삼성테크윈 정밀기기연구소 (Samsung Techwin.)
kingseft@samsung.co.kr
http://myhome.naver.com/kingseft

JMF로 캡쳐 작업을 하기위한 준비사항

1. 인터넷 브라우져 세팅확인

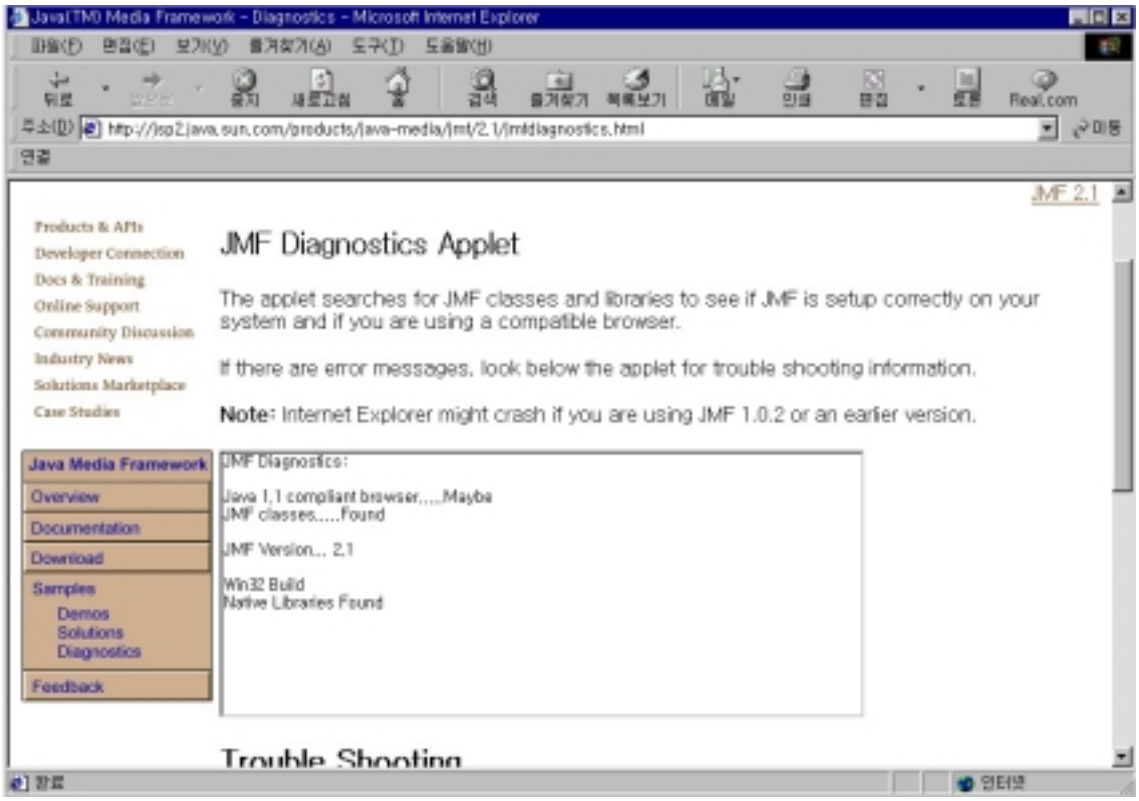
JMF의 설치부분에서도 이미 언급한바가 있지만, 캡쳐 작업을 위해서는 여러분이 각자의 시스템에서 반드시 설정을 해주어야 할 부분이 있다. 먼저 여러분의 인터넷 웹 브라우져가 JMF 세팅 사항을 만족하는지 확인하여 보자. 먼저 다음 웹싸이트를 방문하여 보자.

http://jsp2.java.sun.com/products/java-media/jmf/2.1/jmfdiagnostics.html

위의 웹싸이트를 통하여 여러분은 현재 이용하고 있는 인터넷 브라우져가 JMF 프로그램을 구동하기 위한 설정이 올바르게 되었는지를 알 수 있다. [그림 1]에서 보듯 위의 싸이트에서는 JMFDiagnostics 클래스를 구동시켜서 여러분의 인터넷 브라우져의 세팅사항을 판단한다. JMFDiagnostics 애플릿에 의한 판단 결과는 다음과 같이 나타나게된다.

```
JMF Diagnostics:
Java 1.1 compliant browser ..... Maybe
JMF classes ....Found
JMF Version ....2.1
Win32 Build
Native Libraries Found
```

JMFDiagnostics를 통해 판단된 결과는 여러분의 인터넷 브라우져의 자바 호환여부, JMF classes가 설치되어있는가의 여부와 JMF version, 마지막으로 Native Libraries가 제대로 설치되어있는지를 판단한다. 만약 위의 판단 내용 결과가 여러분이 실험한 결과와 다르다면 JMF 세팅이 잘못된것으로 다음과 같은 문제해결 방법을 취해야한다.



[그림 1] JMF Diagnostics Site 를 통한 인터넷 브라우져의 JMF 세팅확인

JMFDiagnostics 문제해결 방법

1. Browser not compatible:

JMF는 자바 1.1 호환 브라우져를 요구한다. 추천되는 웹 브라우져는 Netscape Navigator 4.06 이상, MS Internet Explorer 4.0 이상, Sun's Hotjava 1.1.x. 이다. 만약 이전 버젼의 브라우져를 사용한다면 최신 버젼의 브라우져로 다시 설치하셔야 한다. 또하나 주의해야 할 점은 실제 이전 버전의 브라우져를 사용한다면, 최신 버젼의 브라우져를 설치한후 JMF를 재설치 해야 한다는 것이다. 이와 같은 과정을 통해서만 JMF를 설치하는 과정에서 Netscape를 인식해서 필요한 파일들도 다시 정리해 주기 때문이다.

2. Classes not found:

JMF classes 및 CLASSPATH 환경 변수에 제대로 설정이 되어있지 않을경우 이와 같은 메시지가 발생한다. CLASSPATH 에서 jmf.jar 를 기록해주었는지 확인한다. 일반적으로

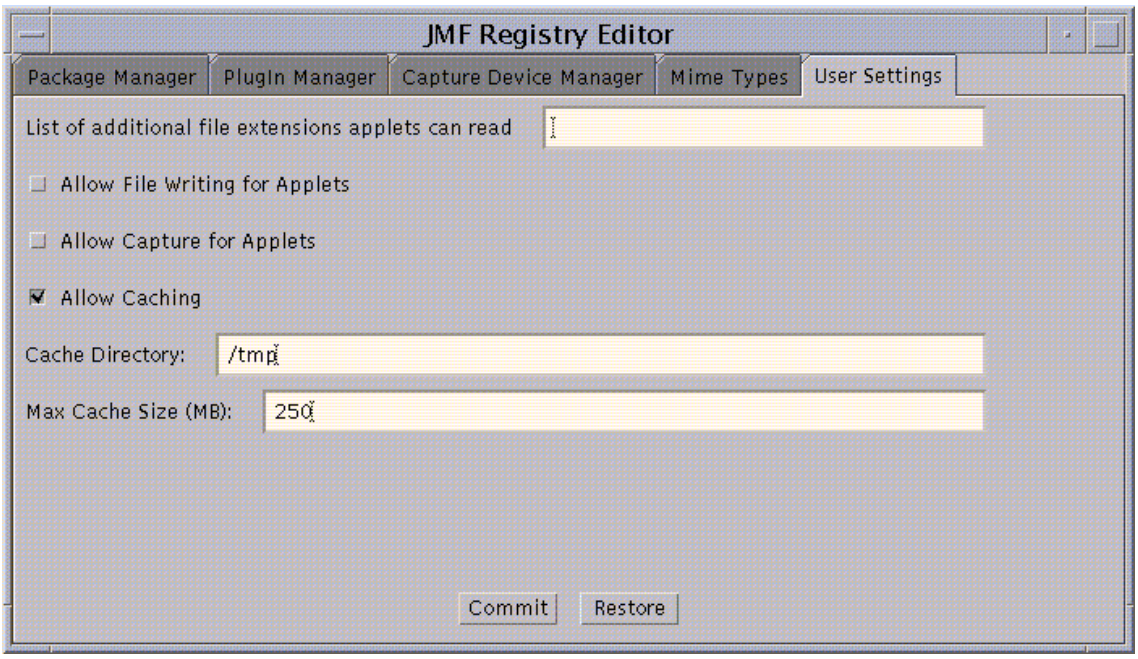
JMF 를 설치하면 이 부분은 자동 세팅되고 컴퓨터를 재부팅하는데 혹시라도 이 부분에서 에러메시지가 발생했다면 , 여러분은 직접 CLASSPATH 를 적어 주어야 한다.

3. Libraries not found:

JMF를 설치후에 설치되는 JMStudio 프로그램이 캡쳐가 안된다고 할때. 가장 흔하게 나타나는 문제점이 바로 이 부분이다. 이 부분에서 에러가 발생하면 캡쳐 프로그램을 수행하는 경우 예측 못할 결과가 발생한다. 만약 윈도우나 솔라시스같은 환경에서 캡쳐 관련 프로그램을 작성한다면 JMF 자체도 역시 Native Version의 JMF를 다운 받아 설치하였을것이다. 다음으로 이 에러가 발생하는 이유는 JMF의 lib 디렉토리를 브라우져가 제대로 확인하지 못해서 NativeCode를 찾지 못하기 때문에 발생한다. 이러한 라이브러리들은 여러분이 설치한 JMF의 하위 디렉토리인 Wlib 밑에 있으며 이 디렉토리를 여러분의 bat 파일에 기록해 주어야 한다. 하지만, 문제점 또다른 하나의 문제점은 만약 IE를 사용하고 있고, 분명히 제대로 설정도 다 해주었는데 왜 안되는가 하는 질문에 답변은 가장 최신 버젼의 IE용 java VM을 다시 다운로드 받아야 한다는 것이다. 다음 싸이트를 통하여 http://www.microsoft.com/java java VM을 다운로드 받아 설치하도록 하자.

2. JMFRegistry 설정바꾸기

JMFRegistry는 JMF 인스톨과정에서 함께 설치되는 JMF 관련 등록정보 편집기이다. 이 프로그램을 통하여 현재 자신의 시스템에 설정되어있는 JMF 관련 Codec정보, Multiplexer 정보, 캡쳐디바이스의 검색과 등록/삭제, 애플릿에서의 캡쳐장치 사용에 관한 허가내용을 부여할 수있다. 우리는 애플릿환경에서도 클라이언트 시스템의 캡쳐기능 지원을 위하여 JMFRegistry를 통하여 기본 설정을 수정해주어야 한다. 먼저 JMFRegistry를 구동한후에 아래와 같은 사항을 선택한다.



[그림 2] JMFRegistry 구동 화면

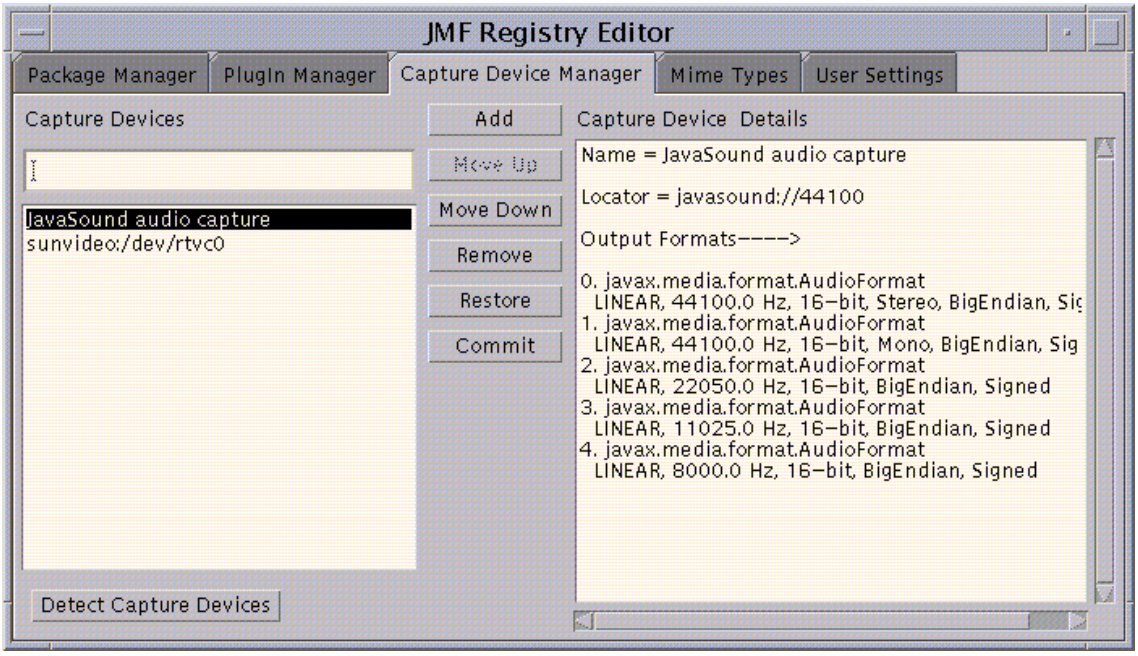
그림에서 보면 기본적으로 Java의 보안설정 문제로 인하여 애플릿에서 파일저장과 애플릿에서의 캡쳐기능에는 디폴트로 선택이 되어있지 않다. 그러므로 우리는 이 부분에서

V Allow File writing for Applets

V Allow Capture for Applets

위의 두 부분을 선택해 주고 Commnit 버튼을 눌러 그 내용을 저장해 주어야 한다. 이와 같은 과정을 통하여 이제 애플릿에서도 네트워크를 통해 수신된 파일을 저장할 수도 있으며, 애플릿상에서 오디오 및 비디오 데이터에 대한 캡쳐 작업을 수행할 수 있게된다.

자. 그러면 JMFRegisty에서 한가지만 더 살펴보고 가자. JMF Registry에서는 자신의 시스템에 현재 설정되어 있는 오디오 및 비디오 캡쳐디바이스 드라이버에 대한 이름과 세부 사항의 정보를 알려준다. 독자의 컴퓨터 시스템에 따라 출력되는 내용이 아래 그림과는 다를 수 있지만, 일반적으로 오디오 캡쳐장치와 비디오 캡쳐 장치가 있다는 전제하에 설명을 하겠다. 먼저 이 부분을 통하여 여러분은 Detect Capture Devices 버튼을 이용하여 JMF 설치과정에서 탐색하여 등록하지 못했던 새로운 캡쳐장치의 디바이스 드라이버를 검색하여 등록할 수 있다. 또한 오디오 및 비디오 캡쳐장치의 디바이스 드라이버를 찾는 순서를 Move Up, Move Down 버튼을 이용하여 지정하는 것이 가능하다. Capture Device Details에서는 캡쳐 디바이스 드라이버에서 제공되는 캡쳐포맷을 나타내어준다. 오디오 디바이스 드라이버의 경우에는 오디오 샘플링레이트, 압축포맷, Endian지정, 부호화의 여부를 나타내며, 비디오의 경우 비디오 포맷의 크기, RGB/YUV 등의 출력양식, 프레임 레이트 정보를 알려준다. 그러므로 JMF 캡쳐 프로그래밍에 앞서서 여러분의 시스템 환경에 적합한 오디오 및 비디오 포맷을 염두해 두고 있어야 한다.



[그림 3] JMF Registry의 캡쳐 디바이스 정보 표시

JMF 데이터 캡쳐작업

Java Sound를 통하여 오디오에 대한 데이터를 얻는 기능이 있었지만, 상위 수준의 오디오 및 비디오 데이터에 대한 캡쳐기능은 JMF 2.x 버전부터 새롭게 지원되었으며, 비디오 카메라와 캡쳐카드 혹은 USB 카메라등을 통하여 비디오 데이터를 획득할 수 있으며, 사운드 카드를 통하여 여러분은 오디오 데이터를 캡쳐하고 저장하며 네트워크를 통한 전송 기능을 구현할 수 있게되었다. 먼저 JMF에서 제공하는 미디어 데이터 캡쳐의 과정을 살펴보자

1. CaptureDeviceManager 클래스를 통하여 캡쳐디바이스 장치의 위치 정보를 얻는다.
2. 캡쳐디바이스 장치에 대하여 CaptureDeviceInfo 객체 정보를 얻는다.
3. CaptureDeviceInfo 객체로부터 MediaLocator 정보를 얻어온후 이를 이용하여 DataSource를 생성한다.
4. DataSource를 이용하여 Player 또는 Processor를 생성한다.
5. 캡쳐 작업을 시작하기 위하여 Player 또는 Processor를 start 시킨다.

JMF에서 캡쳐작업을 수행하기 위해서는 먼저 CaptureDeviceManager 를 통해서 캡쳐 장치에 접근해야한다. 이 클래스는 JMF 에서 이용가능한 모든 캡쳐장치의 등록정보를 관리하는 일종의 registry 역할을 한다. 이 클래스를 통해서 특정한 캡쳐 장치를 직접 조사하거나 혹은 현재 시스템에 장착된 모든 사용가능한 캡쳐 장치의 목록을 얻어 올수도 있다. 이러한 과정을 통하여 획득된 각각의 캡쳐장치들은 각각 CaptureDeviceInfo 객체로서 표현이 된다. 이 과정을 부분적으로 구현된 아래의 코드와 비교하면서 살펴보자. 이 프로그램은 현재 자신의

시스템에 비디오 캡쳐장치가 있는경우 비디오 캡쳐장치에서 어떤 비디오 포맷형식들을 지원하는지를 알려준다.

[프로그램 1] 구동 시스템의 비디오 캡쳐장치의 포맷리스트의 구현부분

```
CaptureDeviceInfo di = null;
Process p = null;
Format[] format = null;

Vector deviceList = CaptureDeviceManager.getDeviceList(null);

if(deviceList.size() > 0 ) {
  System.out.println("캡쳐디바이스 장치를 찾았습니다.");
} else {
  System.out.println("탐색된 캡쳐장치가 없습니다.");
  System.exit(-1);
}

for(int i=0; i< deviceList.size() ; i++ ){
  di = ((CaptureDeviceInfo)deviceList.elementAt(i));
  format = di.getFormats();
  for(int j=0; j < format.length ; j++){
    if( format[i] instanceof VideoFormat){
      Dimension size = ((VideoFormat)format[j]).getSize();
      String encoding = ((VideoFormat)format[j]).getEncoding();
      float frameRate = ((VideoFormat)format[j]).getFrameRate();
      String frame = String.valueOf(frameRate);
      list.add(size.width + " " + size.height + " " + encoding + " " + frame);
    }
  }
}
```

위의 프로그램을 살펴보면 먼저 CaptureDeviceManager.getDeviceList(null); 를 호출하고 벡터 형식으로 반환되는 값을 저장한다. CaptureDeviceManager의 getDeviceList(Format format) 메쏘드는 인자로 주어진 포맷에 맞는 현재 캡쳐가능한 모든 캡쳐 디바이스장치들에 대한 리스트를 반환한다. 만약 인자로서 null을 줌으로서 모든 오디오 및 비디오 캡쳐 장치를 검색하고 리턴값을 반환하게된다.

[구현 1]

```
Vector deviceList = CaptureDeviceManager.getDeviceList(new AudioFormat(“ linear”, 44100, 16,2));
```

[구현 2]

```
Vector deviceList = CaptureDeviceManager.getDeviceList(null);
```

[구현 1]에서는 특정한 오디오 포맷과 샘플링 레이트를 미리 알고있다는 전재하에서 그에 알맞은 캡쳐 디바이스 드라이버를 찾을 것을 명령하는 부분이고, [구현 2]에서는 현재 어떤 캡쳐 디바이스 장치가 있는지에 관계없이 시스템의 모든 캡쳐장치를 찾으라는 명령이다. 이 방법으로 캡쳐디바이스 장치를 찾은경우 주의할 점은 오디오캡쳐장치와 비디오캡쳐장치에 대한 모든 디바이스 드라이버를 검색해서 결과로 리턴하기 때문에 실제로 오디오 캡쳐기능을 이용할지 비디오 캡쳐 기능을 이용할지를 판단하는 부분이 바로 우리에게 주어진다는 점이다. 이제 반환된 배열정보에서 CaptureDeviceInfo 객체로 변환된 정보를 얻어오고, 이 디바이스가 오디오디바이스인지 비디오디바이스인지를 검사해야 한다. 이부분의 검색을 위하여 Format 이라는 클래스를 이용한다. 위의 프로그램에서는 비디오포맷을 선택하였고 오디오 포맷일경우 무시를 하였다. 이제 비디오 포맷에 대한 크기와 프레임레이트 및 부호화타입에 대한 정보를 얻어옴으로서 우리가 시스템에서 사용가능한 비디오 포맷에 대한 정보를 얻는 모든 과정을 마치게 된다.

실제적인 미디어 데이터에 대한 캡쳐 작업수행하기

위에서 CaptureDeviceManager.getDeviceList()를 통하여 캡쳐디바이스장치에 대한 정보를 얻어오는 방법을 확인하였다. 이렇게 획득한 특정한 캡쳐장치로부터 미디어 데이터를 캡쳐하기 위해서는 CaptureDeviceInfo로 변환된 객체를 통해 데이터 소스를 생성해야한다. 다음의 프로그램을 살펴보도록 하자.

[프로그램 2] 캡쳐장치의 획득과 데이터소스의 생성

```
Vector vectorDevices = CaptureDeviceManager.getDeviceList (null);
Vector videoDevice = new Vector();
int nCount = vectorDevices.size();
Format arrFormats[];

    for (int i=0; i<nCount; i++) {
            captureDeviceInfo = (CaptureDeviceInfo)vectorDevices.elementAt(i);
            arrFormats = captureDeviceInfo.getFormats();
```

```
        System.out.println(arrFormats);
        for (int j=0; j<arrFormats.length; j++) {
          if(arrFormats[j] instanceof VideoFormat) {
            videoDeviceName = captureDeviceInfo.getName();
            videoDevice.addElement(captureDeviceInfo);
            break;
          }
        }
      }
      System.out.println(" 비디오 디바이스 이름 : " + videoDeviceName);
      captureDeviceInfo = (CaptureDeviceInfo)videoDevice.elementAt(0);
      try {
        ds = Manager.createDataSource(captureDeviceInfo.getLocator());
        player = Manager.createPlayer(ds);
        System.out.println("플레이어 만들기(데이타소스 받아오기)");
        player.addControllerListener(controllerHandler);
        start();
        System.out.println("캡쳐 시작");
      } catch(Exception ex) {
        ex.printStackTrace();
                }
```

위의 프로그램에서는 획득한 이용한 모든 캡쳐 장치를 먼저 affFormats를 통하여 System.out.println(arrFormats); 의 방법으로 출력을 하였고, arrFormats[j] instanceof VideoFormat 를 통하여 비디오 포맷일경우에만 다음 과정으로 수행한다. 비디오 캡쳐 장치인 경우에는 captureDeviceInfo.getName(); 를 통하여 실제 캡쳐장치의 이름을 얻게된다. 획득한 캡쳐데이터장치로부터의 데이터소스의 생성은 Manager 클래스의 createDataSource() 메쏘드를 통하여 이루어지며, 이때 인자로서 현재 캡쳐디바이스 장치가 있는 위치를 지정해 주기 위하여 위에서 얻은 CaptureDeviceInfo의 getLocator() 메쏘드의 반환값을 이용하였다. 이렇게 생성된 데이터소스는 결국 Player의 생성을 위한 인자로 연결되며, 이 Player를 start 시킴으로서 미디어 데이터의 캡쳐 작업이 실행된다.

User 측면의 Capture Processor 제어하기

캡쳐장치는 일반적으로 일련의 구현된 속성을 지니고 있는데 이러한 속성을 통해 캡쳐 장치를 제어할수 있다. JMF에서는 2개의 컨트롤 타입을 제공하는데 PortControl과 MonitorControl 이다. 이러한 컨트롤들은 캡쳐 DataSource에 대한 getControl 메쏘드를 호

출함으로서 억세스를 할수있다. PortControl은 어떤 데이터가 캡쳐될지에 대한 포트를 선택하게 하며, MonitorControl은 디바이스의 캡쳐 모니터링을 표시하기 위한 수단으로 이용된다. 그러나 이러한 컨트롤보다는 좀더 사용자에게 친근한 컨트롤 즉, 시각적인 컴포넌트(Visual Component)가 기본적으로 제공된다. 이것은 getControlComponent 를 호출함으로서 얻어진다. 이렇게 얻어진 컴포넌트를 우리가 원하는 애플릿 혹은 판넬등에 부착시킴으로서 사용자로 하여금 캡쳐과정을 제어할 수 있는 방법을 제공하게된다. [그림 4]에서는 JMF에서 기본적으로 제공하는 control panel component와 visual component이다. 이러한 컴포넌트를 얻는 방법을 살펴보자.

[그림 4] 캡쳐과정의 Component

[프로그램 3] JMF의 컴포넌트 얻어오기

```
ControllerHandler controllerHandler = new ControllerHandler();
Player player.addControllerListener(controllerHandler);
class ControllerHandler implements ControllerListener {
    public synchronized void controllerUpdate(ControllerEvent ev) {
        if(ev instanceof RealizeCompleteEvent) {
            Component visualCompo;
            Component controlPane;
            visualCompo = player.getVisualComponent();
            controlPane = player.getControlPanelComponent();
            if(visualCompo != null) add("Center", visualCompo);
            if(controlPane != null) add("South", controlPane);
            validate();
          }
       }
     }
    }
```

부분적인 프로그램이지만, 이부분에서 주의할 점은 바로 Player 또는 Processor가 RealizeCompleteEvent를 받은후에 실제로 Component를 얻어올 수 있다는 점이다. 이전 강좌에서도 언급을 했지만, Player, Processor는 각 상태에 따라서 할수 있는 작업과 할 수 없는 작업이 정해져 있다. 다음에는 오디오 데이터를 캡쳐하여 플레이하는 프로그램을 살펴보자. 앞에서도 언급했지만, 오디오 캡쳐를 위해서는 오디오캡쳐장치에 대한 CaptureDeviceInfo 객체 정보를 얻어와야하고, 그 객체 정보로부터 데이터 소스를 획득하여

player를 생성한다. 또 다른 방법으로는 MediaLocator를 이용하여 player를 생성하는 방법을 이용할 수도 있다. 아래의 부분적인 프로그램을 살펴보자.

[프로그램 4] 오디오의 캡쳐와 재생방법

```
Processor p;
String linear = javax.media.format.AudioFormat.LINEAR;
javax.media.format.AudioFormat af
  = new javax.media.format.AudioFormat(linear, 44100, 16, 2);
Vector deviceList = CaptureDeviceManager.getDeviceList(af);

if (deviceList.size() > 0) {
    di = (CaptureDeviceInfo)deviceList.firstElement();
}
else
{
    System.out.println("Could not find a suitable device");
    System.exit(-1);
}
try{
  p  = Manager.createProcessor(di.getLocator());
}catch(IOException e){
}catch(NoPlayerException e){
}
```

위의 프로그램에서는 javax.media.format.AudioFormat.LINEAR 의 형태로 오디오 캡쳐의 형식을 지정하고, javax.media.format.AudioFormat(linear, 44100, 16, 2); 를 통하여 44100 샘플링레이트와 16비트 및 2채널 오디오 포맷을 지정하였다. 이러한 포맷을 미리 지정하고 이렇게 원하는 포맷의 캡쳐디바이스 드라이버가 있는지를 판단한후에 디바이스 장치의 리스트를 얻어온다. 이후 getLocator() 메쏘드를 통하여 캡쳐 장치의 위치정보를 인자로 지정하여 프로세서를 생성하게 된다.

캡쳐된 데이터의 저장과 부호화 포맷지정

위에서 살펴본바와 같이 캡쳐 과정에서는 Player, Processor를 모두 이용할 수 있으나 각각의 제약점이 있는데 이 부분에 대해 알아보도록 하자. 먼저 Player는 캡쳐 작업에 이용할 수 있으나 파일의 저장이나 캡쳐된 데이터의 네트워크를 통한 전송등의 작업을 할 수 없다. Processor는 파일의 저장이나 네트워크의 전송기능을 수행할 수 있으나, 실제로 화면에 보

이는 Player의 Display 부분의 역할을 할 수가 없다. 이러한 각각의 제약점 때문에 몇몇 해결방법이 제시되고 있다. 먼저 캡쳐된 데이터의 저장에 관한 방법부터 알아보자.
캡쳐된 미디어데이터의 저장을 위해서는 Processor가 이용되며, Processor 객체의 출력 데이터소스로부터 미디어데이터를 읽어들여 DataSink를 생성하여 저장에 이용하게된다. 구체적인 저장과정을 아래에 나타내었다.

1. Processor의 getDataOutput 메쏘드를 이용하여 Processor의 출력된 DataSource를 얻는다.
2. Manager.createDataSink 를 이용하여 file writer인 DataSink를 생성한다. 출력된 데이터 소스와 실제 파일을 저장할 위치정보를 담고 있는 MediaLocator를 생성한다.
3. DataSink의 open 함수를 호출하여 실제 저장할 파일을 열게된다.
4. DataSink의 start 함수를 호출하여 파일 기록시작을 준비한다.
5. Processor의 start 함수를 호출하여 캡쳐 작업을 시작한다.
6. EndOfMediaEvent 이벤트나 특정지정시간 혹은 사용자의 stop 이벤트를 기다린다.
7. Processor의 stop 함수를 호출하여 캡쳐작업을 멈춘다.
8. Processor의 close 함수를 호출하여 Processor를 종료시킨다
9. Processor가 완전히 종료되고, DataSink가 EndOfStreamEvent를 전달하게되면 DataSink의 close 함수를 호출하여 DataSink를 종료한다.

살펴본바와 같이 여러단계의 과정을 거쳐서 캡쳐 및 저장이 이루어진다. 위에서는 캡쳐 장치가 오디오인경우를 다루었지만, 비디오의 캡쳐역시 동일한 과정을 통해서 이루어지며, 차이점은 캡쳐 디바이스 리스트를 얻은후 그것이 AudioFormat인지 VideoFormat인지만 구별을 해서 CaptureDeviceInfo 객체를 생성한이후 그 다음의 모든 작업은 동일하게 구성된다. 아래에 위에서 언급한 절차를 실제 프로그램으로 구현한 예를 살펴보자.

[프로그램 5] 캡쳐된 데이터의 파일저장

```
CaptureDeviceInfo di = null;
Processor p = null;
StateHelper sh = null;
Vector devicelist = CaptureDeviceManager.getDeviceList(
    new AudioFormat(AudioFormat.LINEAR, 44100, 16,2));

if(devicelist.size() > 0)
  di = (CaptureDeviceInfo)devicelist.firstElement();
else
  System.exit(-1);
```

```
try{
  p = Manager.createProcessor(di.getLocator());
  sh = new StateHelper(p);
}catch(IOException e){
  System.exit(-1);
}catch(NoProcessorException e){
  System.exit(-1);
}

if(!sh.configure(10000))
  System.exit(-1);

p.setContentDescriptor(new FileTypeDescriptor(FileTypeDescriptor.WAVE));

if(!sh.realize(10000))
  System.exit(-1);

DataSource source = p.getDataOutput();
MediaLocator dest = new MediaLocator("file://result.wav");

DataSink filewriter = null;
try{
  filewriter = Manager.createDataSink(source, dest);
  filewriter.open();
}catch(NoDataSinkException e){
  System.exit(-1);
}catch(IOException e){
  System.exit(-1);
}catch(SecurityException e){
  System.exit(-1);
}

StreamWriterControl swc = (StreamWriterControl)
    p.getControl("javax.media.control.StreamWriterControl");
if(swc!=null)
  swc.setStreamSizeLimit(5000000);
```

```
try{
  filewriter.start();
}catch(IOException e){
  System.exit(-1);
}
sh.playToEndOfMedia(50000);
sh.close();
filewriter.close();
```

위에서 설명하지 못한 부분이 있다. StreamWriterControl 부분과 캡쳐된 데이터의 부호화 부분이다. 먼저 StreamWriterControl에 대해 알아보자. 현재 이용하는 processor가 만약에 StreamWriterControl 을 구현한다면, 이러한 StreamWriterControl을 통하여 파일에 캡쳐된 데이터를 저장할때의 저장 용량을 지정해 줄수가 있다. StreamWriterControl의 setStreamSizeLimit 메쏘드를 이용함으로서 이를 구현할수 있다. 그러면 이번에는 캡쳐된 데이터의 부호화(Encoding) 과정을 살펴보자. Processor를 이용하여 presentation이나 데이터의 전송 또는 파일에 저장하기에 앞서서 캡쳐된 미디어데이터 파일에 대한 트랜스코딩 작업(transcode)를 설정해줄 수 있다. 이러한 미디어 데이터의 부호화과정은 다음과 같다.

1. 캡쳐 장치의 MediaLocator을 얻어오고, Processor를 생성한다.
2. Processor의 configure를 수행한다.
3. Processor가 Configured 상태에 진입된 것을 확인한 후에, getTrackControls 를 호출한다.
4. 원하는 포맷으로 변경하기위한 특정한 트랙을 찾을때까지 각 트랙에 setFormat 함수를 호출한다. (예를들어 IMA4 오디오포맷으로 변환하는 과정등등… .)
5. processor를 realize 시키고, processor의 출력 데이터소스를 이용하여 파일에 적기위한 DataSink를 생성한다.

[프로그램 6] 캡쳐된 데이터의 트랜스코딩 부분

```
if(!sh.configure(10000))
  System.exit(-1);

p.setContentDescriptor(new FileTypeDescriptor(FileTypeDescriptor.WAVE));
TrackControl track[] = p.getTrackControls();
boolean encoding = false;
for(int i=0; i< track.length; i++)
```

```
  try{
    track[l].setFormat(new AudioFormat(AudioFormat.IMA4_MS));
    encoding=true;
  }catch(Exception e){
   track[l].setEnabled(false);
  }
}
if(!encoding){
  sh.close();
  System.exit(-1);
}
if(!sh.realize(10000))
  System.exit(-1);
```

[프로그램 6]에서 살펴보면, processor를 통하여 특정한 파일포맷으로 저장될 것을 명시하는 부분이 있고, 이 processor 내부에 존재하는 각 트랙에 대하여 우리가 원하는 (JMF 에서 지원되는) 포맷으로 변경이 가능하다. 위의 예제에서는 IMA4_MS 압축 방식을 사용하였다. 지원되는 오디오 포맷 및 비디오 포맷에 관해서는 JMF API Guide를 참조하기 바란다.

ProcessorModel 이용하기

이전 예제에서는 오디오와 비디오 캡쳐를 각각 별도의 작업으로 생각하였다. 이번 예제에서는 ProcessorModel을 이용하여 오디오와 비디오 데이터를 캡쳐하고 이를 각각 특정한 음성 및 영상포맷으로 변환하며, 각 트랙을 다시 결합하여 저장하는 과정을 보여준다.
트랙포맷과 Output Content type을 명시하고, ProcessorModel을 이용하여 Processor를 생성하게되면, Processor는 자동적으로 캡쳐데이터소스에 연결된다. 아래의 프로그램을 살펴보자.

[프로그램 7] ProcessorModel을 이용한 캡쳐작업

```
Format formats[] = new Format[2];
formats[0] = new AudioFormat(AudioFormat.IMA4);
format[1] = new VIdeoFormat(VideoFormat.CINEPAK);
FileTypeDescriptor outputtype = new FileTypeDescriptor(FileTypeDescriptor.QUICKTIME);
Processor p = null;

try{
  p = Manager.createRealizedProcessor(new ProcessorModel(formats, outputtype));
```

```
}catch(IOException e){
  System.exit(-1);
}catch(NoProcessorException e){
  System.exit(-1);
}catch(CannotRealizedException e){
  System.exit(-1);
}

DataSource source = p.getDataOutput();
MediaLocator dest = new MediaLocator(file://test.mov");

DataSink filewriter = null;

try{
  filewriter = Manager.createDataSink(source, dest);
  filewriter.open();
}catch(NoDataSinkException e){
  System.exit(-1);
}catch(IOException){
  System.exit(-1);
}catch(SecurityException e){
  System.exit(-1);
}

try{
  filewriter.start();
}catch(IOException e){
  System.exit(-1);
}
p.start();
```

위에서 주의할점은 Processor의 생성시에 Manager를 통하여 Realized된 Processor를 생성했다는 점이다. 그러나 위의 방법은 실제 캡쳐작업 과정에서는 일반적으로 많이 사용되는 방법이 아니다. 실제의 프로그램에서는 오디오 및 비디오 캡쳐작업을 각각의 Processor를 따로두어 서로 별도의 작업으로 진행하는 것이 일반적이다.

캡쳐되는 비디오 데이터의 크기 선택부분

JMF 프로그래밍에서 많은 개발자들이 흔히들 JMF 캡쳐 혹은 데이터 전송시의 느린 속도에 대하여 지적한다. 일반적으로 이러한 문제를 해결하기 위하여 압축률을 높여서 데이터를 전송하기도 하지만, 원천적으로 캡쳐 장치로부터 캡쳐되는 데이터의 크기를 줄일 수 있다면 이러한 전송량의 문제나 압축시간등에서도 좀더 효율적인 프로그래밍을 할 수가 있다. 아래 프로그램을 통하여 캡쳐되는 비디오의 데이터 크기를 바꾸는 부분을 살펴보자.

[프로그램 8]

```
CaptureDeviceInfo di = null;
Process p = null;
Format[] format = null;
Vector deviceList = CaptureDeviceManager.getDeviceList(null);
if(deviceList.size() > 0 ) {
      System.out.println("Now find capture devices..");
} else {
      System.out.println("No Device List");
      System.exit(-1);
}

for(int i=0; i< deviceList.size() ; i++ ){
      di = ((CaptureDeviceInfo)deviceList.elementAt(i));
      format = di.getFormats();
      for(int j=0; j < format.length ; j++){
          if( format[i] instanceof VideoFormat){
              Dimension size = ((VideoFormat)format[j]).getSize();
              String encoding = ((VideoFormat)format[j]).getEncoding();
              float frameRate = ((VideoFormat)format[j]).getFrameRate();
              String frame = String.valueOf(frameRate);
              list.add(size.width + " " + size.height + " " + encoding + " " +
frame);
         }
      }
    }
```

위의 프로그램은 현재 시스템에 장착되어있는 비디오 캡쳐 데이터 장치의 데이터소스를 획득하여, 그 데이터소스의 크기, 부호화상태 및 프레임 레이트 정보를 획득하여 List 에 추

가하는 부분이다. 여러분은 이 부분적인 코드를 여러분들 자신의 코드에 첨가하여 현재 시스템에서 사용가능한 비디오 캡쳐장치의 이미지 지원정보를 얻을 수 있을것이다. 자. 이제 남은 부분은 위에서 특정 비디오 캡쳐크기를 선택했을 경우 이러한 선택을 캡쳐데이터 소스에 반영하는 부분이다.

[프로그램 9]

```
cdi = getCaptureDeviceInfo();
ds = Manager.createDataSource(cdi.getLocator());
formatControls = ((CaptureDevice)ds).getFormatControls();
System.out.println(formatControls.length);
formats = formatControls[0].getSupportedFormats();
System.out.println(formats.length);
format = formats[index];
formatControls[0].setFormat(format);
System.out.println("format-"+format);
System.out.println("format 정보"+(formatControls[0].getFormat()).toString());
```

위의 프로그램은 앞에서 List에 추가된 각 이미지크기를 유저가 선택하면 그 값을 index 값으로 받아들여서 데이터 소스의 포맷 정보를 얻어내고, 이렇게 얻어낸 포맷정보를 현재 이용할 데이터소스에 새로운 포맷으로 지정하는 부분이다. 또한 프로그램의 마지막 부분은 포맷을 String 형태로 변환하여 표현하였다.

구현된 프로그램 예제

[그림 4] 완성 프로그램의 구동모습

[그림 4]에서는 간략하게 구현된 비디오 캡쳐 프로그램의 결과이다. 이 프로그램은 JMF의

비디오 캡쳐 프로그래밍과 Swing으로 외관을 꾸민 예제이다. 프로그래밍 세계의 부록 CD에서 제공되는 샘플 프로그램을 통하여 결과를 확인하기를 바라며, 특히 이 프로그램에서 캡쳐장치의 포맷변경만을 통하여 오디오의 포맷 기능이 가능하며, 또한 앞에서 부분적으로 각각 구현된 캡쳐된 데이터의 저장기능, 데이터의 포맷변환기능, 비디오 캡쳐시의 캡쳐 데이터소스 직접 선택하기등의 기능을 여러분 스스로가 제공되는 샘플 프로그램에 첨가하여 구동시켜보기 바란다.

마치면서

이번 강좌를 통하여 독자들은 JMF에서 새롭게 제공하는 비디오 및 오디오 데이터에 대한 라이브 데이터의 획득기능, 획득된 데이터를 가공하여 특정한 데이터 포맷으로의 저장기능, 포맷 변환과 더불어 비디오 캡쳐시의 캡쳐 정보를 획득하는 방법등에 관하여 살펴보았으며, 오디오와 비디오 데이터를 동시에 캡쳐하는 방법을 ProcessorModel로의 구현과 각각의 독립된 Processor를 두어 구현하는 방법에 관해서도 알아보았다. 다음 강좌에서는 이번강좌에서 누락된 부분들과 RTP 통신의 개념 및 JMF에서 구현된 RTP에 대하여 알아보며, JMF에서 기본적으로 제공되는 JMStudio에서의 RTP 송수신 개념에 대해서도 알아보고자 한다.