<JSTORM>

자바 멀티미디어 JMF

중앙대학교 컴퓨터공학과
자바 동호회
JSTORM
http://www.jstorm.pe.kr

Document Information

Document title:	자바 멀티미디어 JMF - 2
Document file name:	JMF2_심빈구.doc
Revision number:	<1.0>
Issued by:	<심빈구> raica@nownuri.net
	정리 : <윤준호> junoyoon@orgio.net
Issue Date:	<2000/8/25 >
Status:	final

Content Information

Audience	마이크로 소프트웨어 독자
	자바 멀티미디어 파운데이션 클래스(JFC)의 사용방법을 알고 싶어하는 중급 프로그래머
Abstract	JMF 프로그래밍
	- 방송국 프로그램의 뼈대에 해당하는 부분인 미디어의 캡쳐와, RTP를 이용한 미디어의 송수신 기능의 구현
Reference	
Benchmark information	

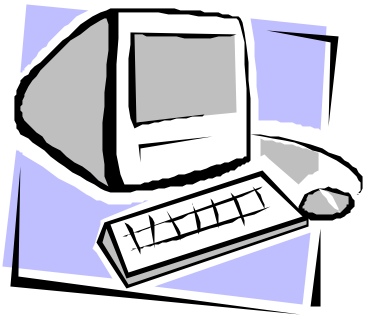
Document Approvals

고문	Signature	date
지위	Signature	date

Revision History

Revision	Date	Author	Description of change

Table of Contents



JMF로 방송제작 프로그램 만들기

저번 연재에서는 JMF의 기본 구조 이해에 중점을 두었었다. 이번 연재에서는 저번 연재에서 예고한대로 본격적인 방송국 프로그램 제작에 들어갈 것이다. 바로, 방송국 프로그램의 뼈대에 해당하는 부분인데, 미디어의 캡쳐와, RTP를 이용한 미디어의 송수신 기능의 구현을 다룰 것이다. 그리고, 다음 연재에서는 이번 연재에서 만든 프로그램에 살을 붙이는 작업을 하게 될 것인데, 방송 선국 장치와 미디어 포맷에 대한 컨트롤 등의 기능을 추가하게 될 것이다.

클래스 설명

이번 회에서 만들려는 방송국 프로그램을 구성하는 클래스들은 <표1>과 같다. MediaPlayer클래스는 저번 연재에서도 포함되었던 MediaPlayer클래스를 좀더 유연하게 확장한 클래스인데, 미디어를 시작시키는 startMedia메소드를 오버로딩(Overloading)하여 여러 가지의 인자로 미디어를 시작시킬 수 있게 하였다.

DisplayFrame은 MediaPlayer클래스에서 생성하는 JFrame형의 displayFrame을 몇 가지 기능을 확장해서 따로 분리해낸 클래스이다. 박스기사에서 설명할 스킨기능과 비쥬얼컴포넌트, 컨트롤컴포넌트의 추가와 삭제의 캡슐화 등을 추가하였다.

클래스		설명
클라이언트	MediaPlayer	클라이언트쪽 주 클래스 ControllerListener와 ReceiveStreamListener를 구현한다.
	DisplayFrame	비쥬얼 컴포넌트와 컨트롤 컴포넌트를 갖는다. setSkin메소드를 통해 스킨을 지원한다.
	RTPDialog	RTP 세션의 정보를 얻는다. RTPUtil을 이용SessionManager을 생성한다
	RTPUtil	SessionManager을 생성한다.

서버	BroadCaster	방송할 데이터 소스를 얻어 방송을 해 준다. setProcessor 모드에 따라 파일 아니면, 캡쳐된 미디어에 대한 Processor을 생성한다. startBroadcast메소드에서는 SendStream 을 생성 방송을 시작시킨다.

표 1 방송국 프로그램을 구성하는 클래스들

이외의 클래스들은 모두 새롭게 추가되어진 클래스들인데, 이번에 구현할 미디어 캡쳐 부분과, 실시간 전송부분을 구현한 클래스들이다.
이번 연재에서는 위의 클래스들 각각을 설명하는 것보다는 각 기능 별로 설명할 예정이다. 각각의 기능별 설명마다 첫 부분에 관련된 클래스를 표시할 테니 전체 코드를 원한다면 해당 클래스를 참고하기 바란다. 이때 클래스 설명부분에 제시하는 부분코드가 포함된 메소드의 앞 뒤 메소드들도 함께 표시될 것이다.
아직 모든 클래스가 만들어 진 게 아니기 때문에 클래스 다이어그램은 생략하였는데, 다음 회에서 모든 클래스들을 포함하는 클래스 다이어그램 통해 더 자세한 설명을 하는 기회를 갖도록 할 것이다.

<표 1>에서 DisplayFrame 이 스킨을 지원한다고 했는데, 이에 대한 사항은 박스기사를 참고하기 바란다.

그림 1 방송국 프로그램 Server와 Client

미디어 캡쳐 맛보기

[관련 : BroadCaster.startBroadcast() -> BroadCaster.setProcessor()]

미디어 캡쳐에 대한 지원은 JMF 2.0 버전에서 새롭게 추가된 기능이다. 음성캡쳐 기능은 그 이전에도 JavaSound를 이용하여 이용할 수 있었다. 하지만 JMF 2.0 버전에서는 음성뿐만 아니라 화상까지도 캡쳐 할 수 있는 기능을 제공해 줌으로써 그 응용범위가 훨씬 다양해 졌다.

그러면, 이러한 캡쳐 기능에 대해 간략히 알아보기로 하자. 미디어 정보를 캡쳐 하려면 먼저 캡쳐가 가능한 장치가 시스템에 설치되어야 할 것이다. 우선 마이크를 설치하자.(이번 연재에서는 화상을 제외한 음성만을 캡쳐해서 방송한다.) 그 후에 해야 할 작업은 바로 설치된 캡쳐 장치를 인식하는 작업일 텐데, 이 작업은 CaptureDeviceManager클래스에 의해서 이루어 진다.

CaptureDeviceManager클래스는 이름에서도 알 수 있듯이 캡쳐 장치에 관한 관리를 맡는 클래스이다. 이 클래스에서는 캡쳐 장치를 등록하는 작업과, 등록된 캡쳐 장치에 대한 정보 제공, 그리고 등록된 캡쳐 장치를 제거하는 작업을 제공한다.

이제, CaptureDeviceManager클래스를 이용하여 캡쳐 장치에 대한 정보를 얻어오자. 그러면 캡쳐장치 인식 작업은 끝나게 된다. 이 정보는 CaptureDeviceInfo라는 클래스에 의해 캡슐화 되어 JMF에서 다루어지게 된다. 이제 CaptureDeviceInfo객체를 얻었다. 하지만, 어떻게 써야 할지 좀 망막할 것이다. 그렇다면 이 클래스가 제공하는 메소드를 한번 살펴보자. 제공하는 메소드라 봐야 몇 개 안 되지만, 그 중에서 아래 메소드가 눈에 들어온 다면 어느 정도 감이 잡힐 것이다.

```
MediaLocator getLocator()
```

바로 MediaLocator객체를 넘겨주는 getLocator이란 메소드 인데, 이 메소드가 넘겨주는 MediaLocator객체는 저번 연재에서 이미 여러번 보았던 클래스 일 것이다. 이 클래스에 대해 다시 한번 간단히 설명하면, 미디어 데이타(스트림)를 캡슐화 하는 클래스인 DataSource객체를 생성하는데 기본적으로 필요한 클래스라고 말할 수 있다. DataSource 객체는 또한 미디어를 플레이하기 위한 Player 객체나, 미디어 조작을 위한 Processor의 객체를 만드는데 기본적으로 필요한 객체인데, 정리하자면 CaptureDeviceInfo 클래스의 getLocator() 메소드에 의해 얻어지는 MediaLocator객체를 사용하여 캡쳐되는 미디어 정보를 플레이하거나 조작하는 것이 가능하게 되는 것이다. 독자들도 느꼈겠지만 파일에서 미디어를 읽어와 플레이하는 것과는 DataSource객체를 얻는 방법만 조금 다를 뿐이지, DataSource객체를 얻고 나서는 캡쳐한 미디어를 다루는 것과, 파일에서 읽어온 미디어를 다루는 것이 하나도 다른 부분이 없다. 바로, 미디어 데이터를 굉장히 잘 추상화 한 DataSource 클래스 덕에 이러한 일이 가능한 것이다.

앞에서는 캡쳐 작업을 순서대로 짚어 보았다. 하지만, 단순히 말로만 진행했기 때문에 대부분의 독자들은 실제로 어떻게 해야 할지는 망막할 것이다. 따라서 이제부터는 예제를 살펴보면서 설명하도록 하겠다.

아래 코드는 CaptureDeviceManager로부터 CaptureDeviceInfo객체를 얻고 최종적으로는 Processor 까지 얻는 곳까지 나타낸 예제이다.

우선 (1)이라고 표시된 곳을 보자. CaptureDeviceManager클래스를 통해 캡쳐 장치정보(CaptureDeviceInfo 객체 리스트)들을 얻는 부분이다. 이때 리턴 되는 캡쳐 장치 정보들은 인수로 넘겨준 Format객체에 적합하게 캡쳐 할 수 있는 장치들이 된다. 만약 인수 값으로 null을 넘겨주게 되면 모든 캡쳐 장치들이 리턴 된다. 따라서 null을 인수로 넘겨줌으로 해서 시스템에 설치된 캡쳐 장치들의 리스트를 얻을 수 있다.

다음으로 (2)라고 표시된 부분은 넘겨진 CaptureDeviceInfo 객체 리스트들 중에서 가장 적합한 객체를 얻는 부분이다. 마지막으로 (3)이라고 표시된 부분에서는 캡쳐된 미디어에 대한 Processor객체를 얻는다. 위에서도 말했지만 이 후부터는 보통의 Processor 사용과 다른 점이 없다.

```
Vector deviceList = CaptureDeviceManager.getDeviceList(new
        AudioFormat(AudioFormat.LINEAR, 44100, 16, 2)); <-- (1)

CaptureDeviceInfo di = null;

if (deviceList.size() > 0)
    di = (CaptureDeviceInfo)deviceList.firstElement();  <– (2)
else{
    System.exit(-1);
}

try {
    this.processor = Manager.createProcessor(di.getLocator()); <-- (3)
} catch(Exception e){
    e.printStackTrace();
}
    return true;
}
return false;
```

RTP 에 대하여

RTP는 그 자체만으로도 상당한 분량이기 때문에, 이번 기회에 설명하는 것은 무리이다. 기본적인 개념만 박스 기사에 다루었으니 참고하길 바라고, 더 자세한 내용은 박스기사에 첨부url 의 문서를 참고하길 바란다. 하지만 박스기사 앞부분에 나오는 payload에 대한 설명과 RTP 세션에 대한 설명은 기본적으로 읽어보고 넘어가야 앞으로 막히지 않을 것이니 그 부분은 꼭 읽어보길 바란다.

Packetizer 과 Depacketizer

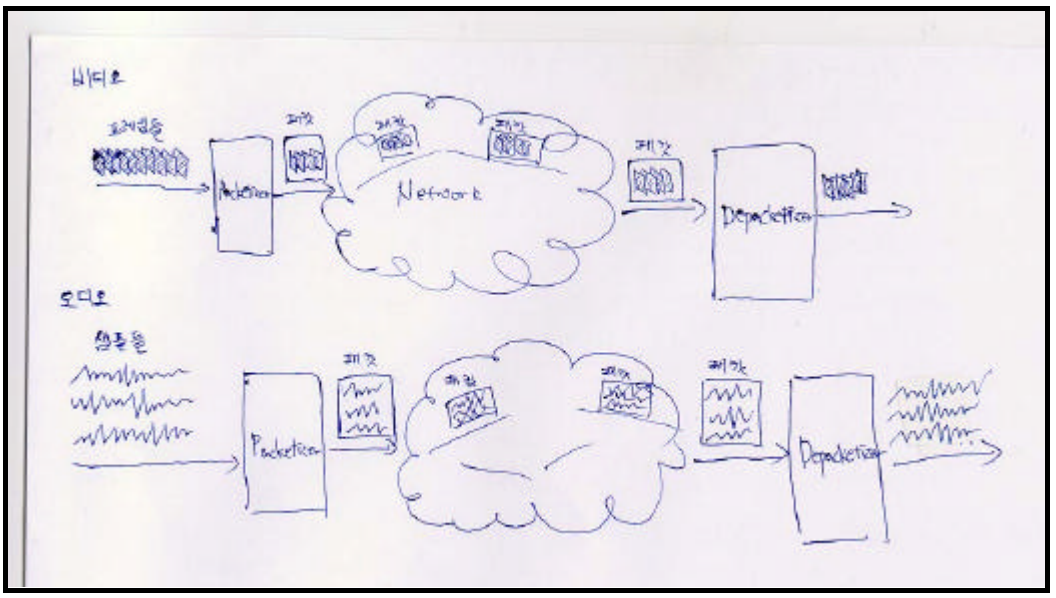


그림 2 Packetizer와 DePacketizer

Packetizer과 Depacketizer는 미디어를 네트웍으로 전송하기 위해 없어서는 안 되는 부분인데, 이름에서도 알 수 있겠지만, Packetizer는 미디어자료를 패킷 형태로 변환하는 일을 하고, Depacketizer는 패킷 형태의 미디어자료를 다시 사용가능한 미디어의 형태로 변환하는 일을 한다. 참고로 패킷이란 네트웍에서 전송의 단위이다.

미디어를 패킷으로 변환하는 과정과 패킷을 다시 미디어로 변환하는 과정은 <그림2>와 같다. 그 과정은 <그림2>에서 보듯이 비디오와 오디오가 서로 다른데, 비디오는 몇개의 프레임 덩어리로 패킷이 구성된다. 이와 반대로 오디오는 여러개의 샘플들이 차례로 그룹화 되어 패킷으로 구성된다.

이러한 모든 작업은 코덱 플러그인에 의해 이루어 진다. 따라서 이러한 코덱을 사용하는 Processor의 프로세싱 과정에서 이루어진다.

RTP를 이용하기

RTP를 사용하여 미디어를 전송하는 방법과 수신하는 방법은 각각 두 가지 방법으로 나뉘어 진다. 우선 전송하는 쪽은 MediaLocator와 DataSink를 이용하여 전송하는 방법과 SessionManager을 직접 사용해서 전송을 하는방법 이렇게 두 가지 방법으로 나눌 수 있다.

수신하는 쪽도 이와 비슷하게 MediaLocator을 이용해 수신하는 방법과 SessionManager을 직접 사용해 수신을 하는 방법 이렇게 두 가지 방법으로 나눌 수 있다. 물론 전송과 수신 모두 첫 번째 방법을 사용하는 것이 훨씬 간단하나 몇 가지 단점이 있다. 자세한 내용은 아래에서 다루도록 할 것이다. 아래에서는 위에서 이야기한 송수신 방법 중 첫 번째 방법들의 쌍을 'RTP를 이용하여 통신하기I'라는 제목으로 그리고, 두 번째 방법들의 쌍을 'RTP를 이용하여 통신하기II'라는 제목으로 설명하게 될 것이다. 하지만 네트웍의 특성상 꼭 이러한 쌍으로 클라이언트와 서버를 구성해야 하는 것은 아니니 알아 두기 바란다.

RTP를 이용하여 통신하기 I

1) 전송

전송하는 방법은 위에서도 이야기했듯이 MediaLocator와 DataSink를 통하여 이루어진다. 아래 코드는 그 예를 보여주는 코드인데, Processor객체를 얻는 과정은 앞 연재에서 이미 다룬 내용이라 중복을 피하기 위해 생략하였다. 따라서 Processor객체는 얻었다고 생각하고 그 후부터 나타낸 코드이다. 우선 살펴 볼 곳이 (1)이라고 표시된 부분이다. 읽혀진 미디어 스트림을 RTP전송에 맞는 형식으로 포맷을 변환하는 부분이다. 따라서 이 후부터는 미디어 스트림을 RTP패킷으로 변경할 수 있게 된다. 이러한 변경 작업은 앞에서 설명한 Packetizer가 수행한다. 여기서는 오디오 전송만을 목적으로 하기 위해서 트랙 중에 비디오 트랙은 제외(setEnabled(false))시켰다. 비디오 전송을 위해서는 반대로 하면 된다. 참고로, RTP 세션은 오디오와 비디오를 한꺼번에 전송하는 멀티미디어의 전송을 지원하지 않는다. 자세한 사항은 박스기사의 'RTP 세션'부분을 보기 바란다.

이제 중요한 부분을 살펴볼 차례이다. (2)라고 표시된 부분을 보자. 바로 MediaLocator객체의 생성자에 넣을 URL 스트링을 생성하는 부분인데, 여기서 생성된 MediaLocator객체는 DataSink에 넘겨져, DataSink가 미디어를 전송할 때 그 목적지로 세팅되어진다. 따라서 (3)에서 생성된 DataSink객체는 DataSource 객체 ds를 url이 rtp://88.3.3.255:22224/audio/1 곳으로 전송하게 된다. (4)부분에서의 start()메소드를 호출함으로써 그러한 전송이 시작되게 된다. 여기서 주소가 88.3.3.255인 것은 멀티캐스트를 위해서이다.

```
processor.configure();
while(processor.getState() != Processor.Configured){}
```

```
processor.setContentDescriptor(
    new ContentDescriptor(ContentDescriptor.RAW));

TrackControl track[] = processor.getTrackControls();
boolean encodingOK = false;

for(int i=0; i < track.length; i++){
 if(!encodingOK && track[i].getFormat() instanceof AudioFormat){
   if( ((FormatControl)track[i]).setFormat(
          new AudioFormat(AudioFormat.GSM_RTP, 8000, 8, 1)) == null) <--(1)
     track[i].setEnabled(false);
   else {
     track[i].setEnabled(true);
     encodingOK = true;
   }
 }else if(track[i].getFormat() instanceof VideoFormat){
   track[i].setEnabled(false);
 }
}

if (encodingOK){
 processor.realize();
 while(processor.getState() != Processor.Realized){}

 DataSource ds = null;

 try{
   ds = processor.getDataOutput();
 } catch(Exception e){
   e.printStackTrace();
   processor.deallocate();
   processor.close();
 }

 try{
   String url = "rtp://255.255.255.255:22224/audio/1";   <--- (2)
   MediaLocator m = new MediaLocator(url);

   DataSink d = Manager.createDataSink(ds, m);           <--- (3)

   d.open();
```

```
    d.start();                                  <-- (4)
      processor.start();
   } catch(Exception e){
    e.printStackTrace();
    setButton(STATE_STOPPED);
   }
  } else{
   processor.deallocate();
   processor.close();
   setButton(STATE_STOPPED);
  }
```

앞에서 이러한 방법이 쉽지만 단점이 있다고 했는데, 위의 코드를 잘 살펴보면 알겠지만, 이 방법으로는 첫 번째 스트림을 보낸 후에 다시 새로운 스트림을 보낼 수 있는 방법이 없다. 그러기 위해선 다시 처음부터 이 작업을 진행하는 번거러움이 있다는게 단점이다. 이때에는 자동적으로 세션이 끊어지게되고, 다시 맺어지는 작업이 수행된다.

2) 수신

RTP 패킷을 수신할 때에도 MediaLocator을 쓰는데, 방법은 위와 비슷하다. 여기에서 쓰인 MediaLocator는 RTP세션을 생성하는데 사용된다. 우선 열려고 하는 RTP세션에 맞는 url로 MediaLoactor을 생성한다. 그런 다음은 파일에서 읽어 플레이하듯이 MediaLocator을 사용하면 된다. 따라서 그 아래 부분(Player객체 생성부분)은 저번 연재에서 보았던 것과 하나도 다르지 않다. 하지만 내부적으로 조금 다른데, Manager는 Player 객체를 바로 생성해 넘겨주는 것이 아니라, RTP세션에서 첫 번째 미디어 스트림이 발견될 때에 Player객체를 넘겨준게 된다.

이제 (3)이라고 표시된 부분을 보자. 바로 player객체를 realize 시키는 부분이다. 저번 연재에서 realize 메소드는 미디어에 대한 정보를 얻고, 미디어 파일 재생에 필요한 자원을 모으는 단계라고 설명하였다. 따라서, 실질적인 데이터를 받기 전에는 realize작업을 끝낼 수 없게 된다. 다시말하면, RTP스트림을 통해 생성된 Player이나 Processor은 실제 RTP데이터 패킷이 도착 하기 전에는 RealizedCompleteEvent를 발생하지 시키지 못한다. 따라서 이러한 상황에서 Manager.createRealizedPlayer() 메소드를 사용하여 Player을 생성하게 된다면 RTP데이터 패킷이 도착하기 전까지는 무한히 Block상태로 있게 된다.

```
String url = "rtp://88.3.3.68:22224/audio/1"; <-- (1)
MediaLocator mrl = new MediaLocator(url);     <– (2)

if(mrl == null){
 System.out.println("지정된 위치의 RTP Session을 찾을 수 없습니다.");
 return;
```

```
}

try{
  player = Manager.createPlayer(mrl);
}catch(NoPlayerException e){
  …
  return;
}catch(MalformedURLException e){
  …
  return;
}catch(IOException e){
  …
  return;
}

player.realize();                    <-- (3)
```

RTP를 이용하여 통신하기 II

SessionManager

앞에서 이 방법을 설명하면서 SessionManager을 직접 사용하여 전송을 한다고 언급했었다. 따라서 먼저 SessionManager에 대해 짚어보고 넘어가도록 하겠다.

SessionManager은 세션에 참여하고 있는 참여자들의 정보와, 전송되어지는 스트림을 관리 하는 클래스이다. 또한 RTCP에 대한 제어기능을 제공한다.

SessionManager은 세션의 참여자들의 정보를 Participant의 객체형태로 갖고 있는다. 이들 참여자들의 정보는 CNAME(canonical name)을 갖는 SDES(source description)을 포함하는 새로운 RTCP패킷이 도착 할 때마다 새로 생성된다.

SessionManager은 또한 전송되어지는 스트림을 ReceiveStream과 SendStream이라는 RTP Stream을 상속받은 두 가지의 스트림을 통해 관리한다. ReceiveStream은 네트웍 반대편의 세션 참여자가 전송하는 스트림을 나타내고, SendStream은 로컬 시스템의 Processor에서 생성되어 넘겨지는 데이터를 네트웍으로 전송하는 스트림을 나타낸다.

ReceiveStream은 SessionManager가 새로운 RTP 데이터를 찾아낼 때마다 자동적으로 생성된다. 이와는 반대로 SendStream은 직접 생성해 주어야 하는데, SessionManager클래스의 createSendStream 메소드를 이용해 생성할 수 있다.

<리스트 : RTPUtil>는 SessionManager를 생성해주는 부분의 코드이다. RTPUtil이란 클래스는 JMF API Guide뒷 부분에 전체 코드가 실려 있는 클래스이다. 실제로 JMStudio에서 RTP전송과 수신에 관계된 부분은 모두 RTPUtil을 이용하리 만큼 JMStudio에서는 많이 사용되는 클래스이다. 하지만 JMStudio에 맞추려다 보니 이름답지 않게 다른 프로그램

에서는 사용할 수 없게 되었는데, <리스트 : RTPUtil>에서는 SessionManager를 만들어주는 static 메소드만 남기고 편집하여 다른 곳에서도 사용할 수 있게 만들었다. 이 메소드에서는 SessionManager을 생성하여 초기화 시키고((1)) 세션을 시작시킨 뒤 ((2)) 넘겨주는 일을 한다.

리스트 : RTPUtil

```
public class RTPUtil{

 public static SessionManager createManager(String address,
                                            String sport, String sttl)
 {
   return createManager(address,
                new Integer(sport).intValue(), new Integer(sttl).intValue());
 }

 public static SessionManager createManager(String address,
                                            int port, int ttl)
 {
   SessionManager mgr = (SessionManager)new
                        com.sun.media.rtp.RTPSessionMgr();
   if (mgr == null) return null;

   mgr.addFormat(new AudioFormat(AudioFormat.DVI_RTP,
                              44100,  4, 1), 18);

   String cname = mgr.generateCNAME();
   String username = null;

   try {
      username = System.getProperty("user.name");
   } catch (SecurityException e){
      username = "jmf-user";
   }

   SessionAddress localaddr = new SessionAddress();

   try{
     InetAddress destaddr = InetAddress.getByName(address);
     SessionAddress sessaddr = new SessionAddress(destaddr,
                                          port, destaddr, port + 1);
      SourceDescription[] userdesclist= new SourceDescription[]
```

```
        {
            new SourceDescription(SourceDescription
                                  .SOURCE_DESC_EMAIL,
                                  "jmf-user@sun.com",1, false),

            new SourceDescription(SourceDescription
                                  .SOURCE_DESC_CNAME,
                                  cname, 1 , false),

            new SourceDescription(SourceDescription
                                  .SOURCE_DESC_TOOL,
                                  "JMF RTP Player v2.0",1, false)
        };

        mgr.initSession(localaddr,                  <---- (1)
                        userdesclist, 0.05, 0.25);

        mgr.startSession(sessaddr,ttl,null);      <---- (2)
      } catch (Exception e) {
        e.printStackTrace();
        System.err.println(e.getMessage());
        return null;
      }
      return mgr;
    }
}
```

1) 전송

[관련 : Broadcaster.startBroadcast()]

아래코드 역시 'RTP를 이용하여 통신하기 I'에서처럼 프로세서를 얻은 뒤부터의 코드이다. 코드의 앞부분은 RTP전송에 맞게 프로세싱 하는 부분이다. 여기서도 역시 Audio 필드만을 남긴다. 그런 뒤에는 RTPUtil을 이용해 SessionManager객체를 얻는다.

다음으로는 SessionManager객체를 통해 SendStream을 생성한다((2)). 이때 두 번째로 넘겨주는 인자는 첫 번째 인자로 넘겨진 DataSource객체가 갖고 있는 스트림들 중에서 사용할 스트림의 인덱스이다. 하지만 넘겨진 DataSource객체에는 오디오트랙만이 존재하니까 인덱스를 0으로 지정해 주었다.

SendStream을 생성하고 난 다음에는 SendStream과 Processor을 각각의 start메소드를 사용해 시작시키면 미디어는 전송이 시작된다.

```
processor.configure();
while(processor.getState() != Processor.Configured){}
processor.setContentDescriptor( new
                        ContentDescriptor(ContentDescriptor.RAW));
TrackControl track[] = processor.getTrackControls();
boolean encodingOK = false;
for(int i=0; i < track.length; i++){
  if(!encodingOK && track[i].getFormat() instanceof AudioFormat){
    if( ((FormatControl)track[i]).setFormat(
                new AudioFormat(AudioFormat.GSM_RTP, 8000, 8, 1)) == null)
      track[i].setEnabled(false);
    else {
      track[i].setEnabled(true);
      encodingOK = true;
    }
  }else if(track[i].getFormat() instanceof VideoFormat){
    track[i].setEnabled(false);
  }
}
if (encodingOK){
  processor.realize();
  while(processor.getState() != Processor.Realized){}

  DataSource ds = null;

  try{
    ds = processor.getDataOutput();
  } catch(Exception e){
    e.printStackTrace();
    processor.deallocate();
    processor.close();
    setButton(STATE_STOPPED);
  }

  try{
    manager = RTPUtil.createManager("88.3.3.63", "22224", "1");   <---- (1)

    if(manager == null){
      System.out.println("12312");
      System.exit(0);
    }
```

```
    SendStream sendStream = manager.createSendStream(ds, 0);      <----- (2)
    sendStream.start();
    processor.start();
   } catch(Exception e){
    e.printStackTrace();
    setButton(STATE_STOPPED);
   }
  } else{
   processor.deallocate();
   processor.close();
   setButton(STATE_STOPPED);
  }
```

2) 수신

[관련 : RTPDialog.open() -> RTPDialog.openSession(), MediaPlayer.update()]

수신하는 쪽 역시 SessionManager을 사용한다. 수신은 ReceiveStream을 이용하는데, ReceiveStream은 앞에서도 설명했듯이 새롭게 도착하는 RTP데이터가 발견될 때마다 SessionManager가 생성한다. 이제 이러한 변화(ReceiveStream의 생성)를 알아내는 일을 해야 하는데, 바로 ReceiveStreamListener을 이용해 알아낼 수 있다. ReceiveStreamListener은 update() 메소드 하나만을 갖는다. 이때 넘겨지는 이벤트 객체는 ReceiveStreamEvent에서 상속받은 몇 가지의 이벤트 객체가 될 수 있는데, 그중에서 우리가 주목해야할 이벤트 객체는 새로운 ReceiveStream이 생성되면 발생하는 NewReceiveStreamEvent 객체이다. 우선 (1)처럼 instanceof 연산자를 통해 원하는 이벤트인지를 검사한다.

그런 다음에는 (2)에서 보듯이 이벤트에서 ReceiveStream객체를 얻어낸다. 마지막으로 (3)에서 보듯이 DataSource객체를 ReceiveStream에서 얻어낸다. 그 후에는 얻어진 DataSource객체를 이용해 재생하면 된다.

```
public void update( ReceiveStreamEvent event)
{
  SessionManager source = (SessionManager)event.getSource();
  System.out.println(event.toString() );

  if (event instanceof NewReceiveStreamEvent)   <------ (1)
  {
    String cname = "Online Broadcast";
    ReceiveStream stream = null;

    try
    {
```

```
        stream =((NewReceiveStreamEvent)event)     <------ (2)
            .getReceiveStream();

        Participant part = stream.getParticipant();
        if (part != null) cname = part.getCNAME();
        // get a handle over the ReceiveStream datasource
        DataSource dsource = stream.getDataSource();   <--- (3)

        this.startMedia(dsource);
      } catch (Exception e) {
        System.err.println("NewReceiveStreamEvent exception "
                + e.getMessage());
        return;
      }
    }
  }
```

이번회를 마치며...

이번회를 통하여 다룬 내용은 저번 회보다는 좀더 실전쪽에 가까운 내용이라, 독자들이 좀더 친근하게 접근 할 수 있었을 것이다. 몇 회 안 되는 분량이지만 회를 더해 가면서 JMF쪽 보다는 JMF를 이용한 실전 프로그래밍 쪽에 중점을 두어 다룰 예정이다. 따라서 다음 회에는 이번회에 만든 프로그램을 좀더 방송국답게 만드는 쪽에 많은 지면이 할당 될것이다.

Sun의 archives 사이트를 활용하자.

Sun의 archives 사이트란 분야별로 나뉘어져, 주로 의견 교환이나 Q&A 게시판 같은 형태로 운영되는 사이트이다. 이 사이트에는 매일 수많은 글들이 세계 각국의 Java 개발자들에 의해 올라오기 때문에 그, 자료의 방대함은 이루 말할 수 없다.

필자는 JMF를 처음 접했을 때 그, 자료의 빈약함 때문에 많은 고생을 했었는데, 무엇이든 새로운 개념을 접할 때에는 필자와 같은 고충을 겪게 되는 일이 다반사다. 물론, 자료의 빈약함이란 한글뿐 아니라 영어로 된 자료까지 포함해서 말한 것이다.

이러한 경우에 필자는 이 사이트를 가보라고 적극 권하고 싶다. 이 사이트의 URL은 아래와 같다.

http://archives.java.sun.com

JFrame에 Skin 입히기

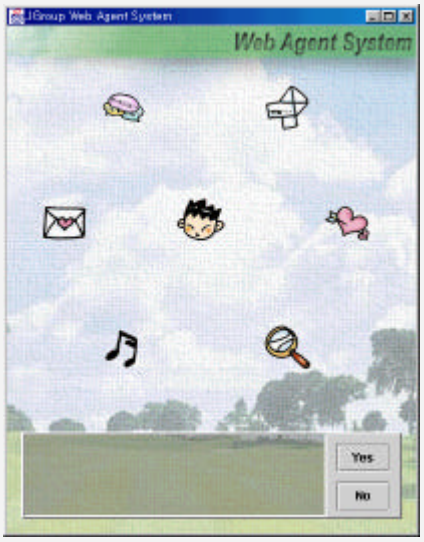


그림 3 스킨을 입힌 JFrame

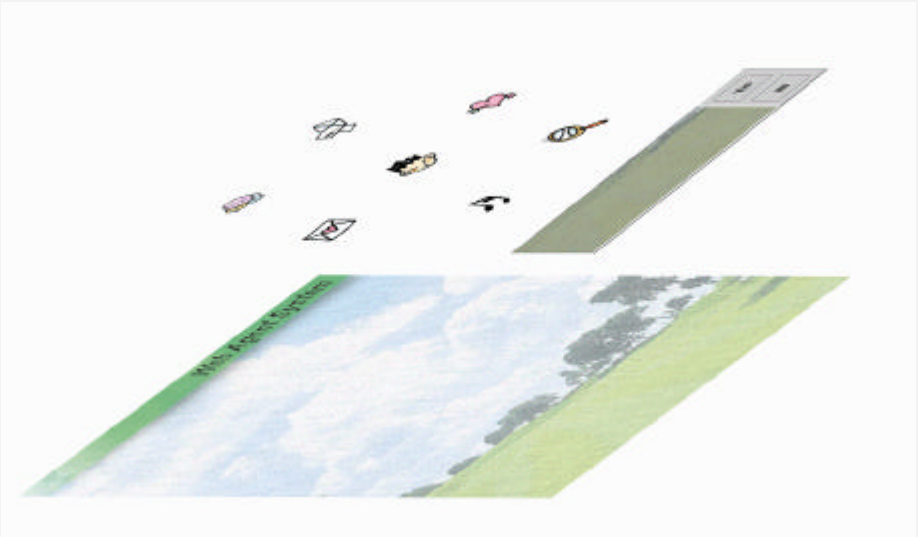


그림 4 <그림3>을 Layer별로 나눈 그림

이 박스기사에서는 Winamp 라는 프로그램을 사용해본 사람이라면 대부분 알고 있는 Skin 이라는 개념을 Swing을 이용하여 적용해 보려고 한다. 일단 JLayeredPane을 조금이라도 알고 있는 독자라면 쉽게 이해할 수 있을 것이다. 혹시 잘 모르고 있는 독자라도, 그다지 어려운 개념은 아니니, 포기하지 말기를 바란다.

우선 JLayeredPane에 대해 잘 모르는 독자를 위해 간단히 설명하고 넘어가겠다. JLayeredPane는 JFC/Swing 컨테이너에 깊이를 제공해 주는 패널이다. 다시 말하면 하나의 컨테이너에 여러개의 패널(JLayeredPane)들을 순서대로 쌓아서 추가할 수 있게 해준다. 이때 각각의 패널들은 고유의 깊이값을 갖게 된다. 물론 깊이 값을 지정해 새로운 패널을 특정 위치에 끼워 넣는것도 가능하다.

아래 코드는 <그림3>에 보이는 프레임을 구성하는 기본적인 코드이다. 코드에 (2)라고 표시된 부분은 그 앞 부분 코드에서 만들어진 스킨 이미지를 갖고 있는 JLabel을 JLayeredPanel에 추가하는 부분이다. 이때 주목해야 할 곳이 주어지는 깊이 값인데, 이 값을 가장 낮은 값(new Integer(Integer.MIN_VALUE))으로 해줌으로써, 스킨을 가장 깊은 곳에다 위치시킨다. 그래야 스킨 밑에 가려서 안 보이는 컴포넌트가 생기지 않기 때문이다.

이제 스킨은 깔아 두었으니 그 위에 컴포넌트를 추가시키는 일만 남았는데, 이때는 한가지 주의해야 할 것이 있다. 바로 스킨 위에 추가되는 컴포넌트에 의해 스킨이 보이지 않게 되는 점인데, 그 해결방법은 간단하다. 추가되는 컴포넌트들의 속성을 투명하게 해 주면 된다. (1)이라고 표시된 곳 들이 바로 속성을 투명하게 바꾸어 주는 부분이다.

```
public void makeUI(){
  JPanel centerPanel=new JPanel();
  JLabel backlabel=new JLabel("");
  final ImageIcon m_image=new ImageIcon
          ("c:\\jagent\\resource\\image\\agent\\backImage.gif");
  final int winc = m_image.getIconWidth();
  final int hinc = m_image.getIconHeight();


  centerPanel.setOpaque(false);                          //<----- (1)
  getContentPane().setLayout(new BorderLayout());
  getContentPane().add(centerPanel, BorderLayout.CENTER);

  ((JPanel)getContentPane()).setOpaque(false);    //<----- (1)

  if (m_image.getIconWidth() > 0 && m_image.getIconHeight() > 0){
    backlabel = new JLabel() {
      public void paintComponent(Graphics g) {
        m_image.paintIcon(this,g,0,0);
      }
      public Dimension getPreferredSize() {
        return new Dimension(super.getSize());
      }
```

```
            public Dimension getMinimumSize() {
              return getPreferredSize();
            }
          };
        }
        getLayeredPane().add(backlabel, new Integer(Integer.MIN_VALUE));// <--(2)
        backlabel.setBounds(0,0,5000,5000);
      }
```

소스에 나오진 않았지만. 위에서 보이는 7가지의 아이콘들은 실제로 JPanel에 에 추가된 그림들인데, 이때에도 역시 각각의 패널들의 속성을 투명으로 주었다. 따라서 그 위에 추가된 그림만이 보이게 되는 것이다.

<그림3>좀더 관심 있게 본 독자라면 아래 부분에 있는 창에 대한 궁금증이 생길 것이다. 그 창은 투명한 것도 아니고, 그렇다고 불투명 한 것도 아니다. 정확히 말하자면 반투명 하다고 해야 할 것이다. 그렇게 하기 위해선 패널에 투명 속성을 주는 것 이외에 한가지 더 해야할 작업이 있다. 바로 배경색을 반투명하게 만들어 주는 일인데, 그러기 위해선 배경색을 지정해 줄 때 RGB값 이외에 추가로 Alpha값을 지정해 주면 된다. 아래 코드는 위에 보이는 창의 배경색에 Red, Green, Blue, Alpha값을 적용시키는 코드이다.

```
showArea.setBackground(new Color(70, 70, 70,50));
```

참고로, Alpha값은 낮을수록 더욱 투명해 지고, 값이 커질수록 더욱 불투명해 지게 된다. 따라서, Alpha값을 지정하지 않고 Color객체를 생성한다면 Alpha값은 기본적으로 255라는 값을 갖게 된다.

실제로 이러한 특성을 이용해 훨씬 더 보기 좋은 GUI를 만들 수 있으니, 여러 가지로 적용시켜 여러분만의 멋진 UI를 만들어 보길 바란다.

IETF RFC 1889 RTP (Real-time Transport Protocol)

RTP는 네트웍 상에서 실시간 전송이 요구되는 상황에 그 해결 방안을 제공해 주는 프로토콜이다. 실시간 전송이 요구되는 상황이란 화상 회의나, 현재 우리가 만들고 있는 인터넷 방송 또는, VOD써비스 등이 될 것이다. 이때 주고받는 데이터는 주로, 비디오나 오디오같은 멀티미디어 정보가 되는데, 실제로 RTP로 전송되는 데이터 패킷 내에는 Payload 타입(PT 필드)이을 지정해 주는 필드가 있어 전송되는 멀티 미디어 자료의 포맷을 표시해 줄 수 있다. 참고로 표1은 RFC 1890에 정의된 Payload 타입이다.

Payload Type	Endocing-Name	a - 오디오 v - 비디오
0	PCMU	a
1	1016	a
2	G721	a
3	GSM	a
4	unsigned	a
5	DVI4	a
6	DVI4	a
7	LPC	a
8	PCMA	a
9	G722	a
10	L16	a
11	L16	a
12-13	unsigned	a
14	MPA	a
15	G728	a
16-23	unsigned	a
24	unsigned	v
25	CelB	v
26	JPEG	v
27	unsigned	v
28	nv	v
29-30	unsigned	v
31	H261	v
32	MPV	v
33	MP2T	av
34-71	unsigned	?
72-76	reserved	n/a
77-95	unsigned	?
96-127	dynamic	?

표 2 RFC 1890에 정의된 Payload 타입

RTP의 Base 프로토콜 선정

실시간 전송 시에는 많이 지연된 테이터를 복구하는 작업보다는 잃어버린 데이터를 복구하는 작업이 훨씬 쉬울 뿐 아니라 실시간이라는 특성을 더 많이 살려 줄 수 있다. 다시 말하면 실시간 전송에서는 전송된 데이터의 질보다는 전송에 걸리는 시간에 더 중점을 둔다고 할 수 있다. 따라서, 정적인 데이터를 처리하는 프로토콜은 스트리밍 미디어를 처리하는 데에는 맞지 않는다. 정적인 데이터 처리는 모든 데이터가 목적지에 도착한 후 처리하는 게 일반적이기 때문이다. 참고로 스트리밍 미디어란 단어는 네트웍에서 실시간으로 전송되어 지고 있는 미디어를 말한다.

TCP는 송신측과 수신측 간에 연결을 맺고 전송을 하는데 전송되어 지는 데이터의 질을 보장 할 수 있다는 장점이 있다. 하지만 이러한 장점 때문에 전송 속도 면에서 손해를 보아야 한다. 중간에 패킷이 하나라도 없어지면 다시 전송을 해줄 때까지 기다리기 때문이다.

이러한 이유 때문에 TCP는 RTP에 적용되기는 부적합하다. 따라서, TCP보다 좀 떨어지는 프로토콜이 스트리밍 미디어의 전송에 쓰인다. 그중 하나가 널리 쓰이고 있는 UDP(User Datagram Protocol)이다. UDP는 TCP와는 대조적으로 중간에 패킷을 잃어버릴 수도 있다. 또한, 도착한 패킷들이 전송된 순서대로 도착하리라는 보장도 없다. 따라서 클라이언트 쪽에서는 손실된 데이터나, 중복된 데이터, 그리고 순서에 맞지 않은 데이터들을 복구할 수 있어야 한다. RTP는 이러한 UDP의 윗단에 붙는 프로토콜이다.

유니캐스트 VS 멀티캐스트

RTP는 이러한 UDP위에서 사용되는 프로토콜이다. RTP는 유니캐스트전송이나, 멀티캐스트 전송 두 가지에서 쓰인다. 유니캐스트전송이란, 하나의 목적지에 데이터를 전송하는 것을 말한다. 따라서 3곳의 목적지에 유니캐스트 전송을 한다면 모든 패킷들의 3개씩 만들어져서 각각의 목적지에 도착하게 된다. 그와는 반대로 멀티캐스트 전송에서는, 송신쪽에서는 오직 하나의 패킷을 만들어 뿌리게 된다. 그러면 그 패킷은 네트웍을 통해 여러개의 목적지에 도착하게 된다. 따라서, 목적지가 많아진다면 멀티캐스트 전송 방법의 훨씬 효율적이게 된다. 멀티캐스티 전송은 IP(Internet Protocol)에서 기본적으로 제공해 준다.

RTP 서비스와 RTCP(Real-time Transport Control Protocol)

RTP는 전송된 데이터의 타입, 또 패킷들을 어떤 순서로 맞추어 재생되어야 하는지?, 그리고 서로 다른 미디어 스트림을 동기화 하는 것 을 가능하게 해 준다.

RTP 데이터 패킷은 실제로 전송된 순서대로 도착할지 보장하지 못한다. 심지어 모든 패킷이 제대로 도착할 지도 보장하지 못한다. RTP자체적으로는 이와 같이 서비스의 질에 대해서 아무런 방법을 제공해 주지 못한다. 때문에, RTCP(Real-time Transport Control Protocol)라는 프로토콜이 RTP에 추가되어 진다. RTCP는 RTP전송의 조정이나, 식별작업을 맡는다.

RTP 세션

RTP를 이용해 통신하는 애플리케이션 쌍(송신측 + 수신측)을 RTP 세션이라 하는데, 이때 세션은 한쌍의 주소로 구분된다(RTP 주소 + RTCP 주소).

보통 미디어의 타입 하나에 하나의 세션이 생성되는데, 오디오와 비디오 두가지 미디어를 전송하는 서버와 클라이언트 사이에는 오디오를 전송하는 세션과, 비디오를 전송하는 세션 이렇게 두 개의 세션으로 연결되어 진다.

RTP 패킷 구조

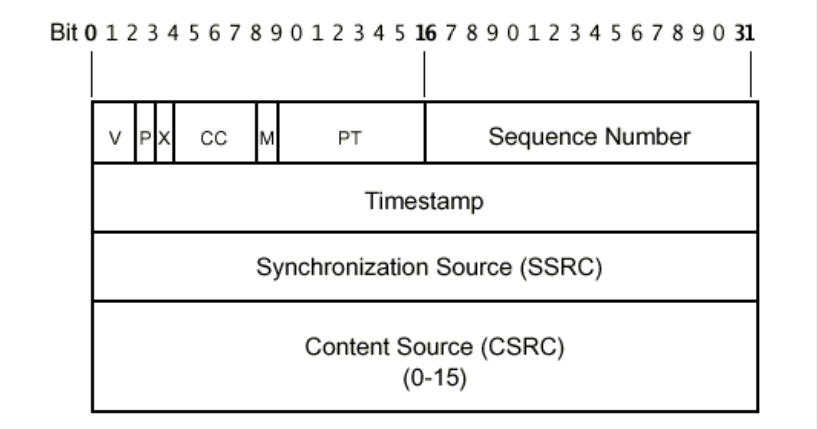


그림 5 RTP 데이터 패킷

PT : payload 타입
Timestamp : 32bit 첫 번째 바이트가 샘플링된 순간, 만약 여러개의 패킷이 동일한
SSRC : 32bit 동일한 믹서(패킷을 생성한곳)에서 생성되었는지를 식별한다. 이 값은 랜덤하게 만들어진다. 같은 RTP 세션에서 전송되어지는 패킷들은 모두 동일한 SSRC값을 갖는다.
CSRC : 32bit씩 실제 데이터 부분

아래는 IETF 사이트의 RTP관련 문서와 payload 관련 문서의 링크이다. 좀더 자세한 사항을 원하는 독자들은 아래 문서를 참고하기 바란다.

RTP - http://www.ietf.org/rfc/rfc1889.txt
Payload - http://www.ietf.org/rfc/rfc1890.txt