제 6 강좌 : RTP 미디어 데이터의 전송

지난 강좌에서는 JMF 2.x 버전에서 가장 큰 주목을 받고 있는 라이브 미디어 데이터의 RTP 를 이용한 실시간 전송 및 수신기능에 대하여 알아보았다. 이를 위하여 실시간 영상 음성 데이터의 송수신을 위한 RTP 의 개념과 헤더 및 데이터 구조에 관하여 논의하였으며, 실제 JMF 에서의 구현부분 및 각 이벤트 처리의 방법에 관하여 논의하였다. 이번 강좌에서는 앞서 언급했던 RTP 관련 패키지들을 이용하기 위하여 프로세서의 설정방법과 출력데이터를 획득하는 방법, 데이터 전송을 위한 RTP 포맷변환 작업, 통신의 구현 방법상의 문제들에 관하여 살펴보며, 데이터를 전송하기 위한 실제적인 프로그램 작업들에 관하여 알아보도록 한다. 또한 웹브라우져를 통하여 애플릿을 이용한 통신과정에서 나타나는 서명된 애플릿을 사용하는 방법을 구현하여 본다. 강좌에 대한 문의나 관련 프로그램은 아래의 홈페이지를 참조하기 바란다.

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

RTP 미디어 데이터의 전송방법

이번 강좌부터는 직접적인 미디어 데이터들을 RTP 프로토콜을 이용하여 전송하는 방법에 관하여 논의한다. RTP 스트림 데이터를 전송하기 위해서는 Processor 를를 이용하여 RTP 포맷으로 부호화된 데이터소스를 생성해내고, 데이터 전송을 주관하는 Session Manager 혹은 DataSink 를 생성해야한다.

Processor 로의 입력은 하드디스크에 저장된 데이터 혹은 원격 네트워크에 연결된 데이터 및 실시간으로 캡쳐되는 데이터를 이용할 수 있다. 하드디스크등에 저장된 데이터는 MediaLocator 를 통하여 Processor 를 생성시에 파일의 위치와 이용 프로토콜을 명시할 수 있다. 캡쳐 작업을 통하여 획득된 데이터에 관해서는 DataSource 를 이용하여 Processor 의 입력으로 이용될 수 있다.

JMF 2.1 버전까지에서는 RTP 스트림의 전송을 위한 2 가지의 방법이 제시되었다. 첫번째는 RTP 세션의 각종 파라미터를 갖는 MediaLocator 를 이용하여 Manager 클래스의 createDataSink 를 호출함으로서 RTP DataSink 를 생성하는 것이다.

두번째 구현 방법은 Session Manager 를 이용하여 전송할 데이터를 위한 send 스트림을 생성하고 전송을 조절하는 방법이다.

그러나 여기서 주의할 부분이 있다. 만약에 여러분들이 MediaLocator 를 이용하여 RTP DataSink 를 생성한다면, 여러분이 전송하는 데이터 소스에서 여러가지 존재하는 미디어 데이터 스트림중에서 단지 첫번째 스트림만 전송을 할 수 있다는 점이다. 만약 하나의 세션을 통하여 여러 개의 RTP 스트림을 전송하려거나 혹은 세션에 대한 통계정보에 대한 모니터링을 하고자 할때는 직접적으로 Session Manager 를 이용해야 한다. 하지만 위의 두 가지 방법중 어떤 방법을 이용한다 하더라도 RTP 스트림을 전송하기위해서는 먼저 실제로 여러분이 전송하고자 하는 스트림 데이터를 표현해주는 DataSource 및 이를 이용하는 Processor 를 생성해야 한다. 두번째로는 RTP 전송포맷으로 부호화된 데이터 출력을 얻을 수 있도록 Processor 의 특성을 조정해야 하며, 마지막으로 이렇게 조정된 Processor 를 통하여 출력되는 데이터를 DataSource 의 형태로 획득하는 단계를 거쳐야 한다.

Processor 의 환경 설정하기

RTP 로 부호화된 데이터를 생성하기위한 Processor 를 만들기 위해서는 전송하려는 미디어 스트림에 존재하는 모든 각각의 트랙들에 대하여 RTP 에서 정한 포맷으로 설정을 해주어야 하며, Output Content Descriptor 를 지정해 주어야 한다. 이러한 트랙의 포맷들은 각각의 트랙들에 대하여 TrackControl 얻음으로서 개별적인 세팅을 할 수 있고, setFormat 을 통하여 RTP 포맷을 지정해준다. 또한 프로그램의 가독 능력을 높이기 위하여 " AudioFormat.GSM_RTP " 와 같은 스트링형태로 부호화 포맷을 명시해줄 수도 있다.

프로세서는 시스템에 설치된 플러그인을 통하여 앞에서 세팅한 포맷의 정보를 얻으려 시도하며, 적절한 플러그인이 설치되지 않았다면 특정 RTP 포맷은 지원이 될 수 없으며, UnSupportedFormatException 예외를 발생하게된다. 또한 JMF 에서의 RTP 구현에서 주의할점은 RTP 규약에서 명시한 모든 미디어 스트림 데이터를 전부 지원하지는 않는다는 점이다. 독자들은 반드시 JMF RTP 문서를 살펴봐주기 바란다. 프로세서의 출력 포맷은 setOutputContentDescriptor 메쏘드를 통하여 설정할 수 있다. 만약 특별한 멀티플렉싱이 요구되지 않는다면 , 이러한 Output Content Descriptor 는 "ContentDescriptor.RAW" 로 설정할 수 있으며, 반드시 주의할점은 오디오 및 비디오 데이터 스트림은 하나의 출력 스트림으로 합쳐질 수 없다는 점이다. 만약 프로세서내에 존재하는 트랙들이 서로 다른 데이터 타입을 지닌다면, 각각의 미디어 스트림들은 각각 별도의 RTP 세션으로 전송되어진다. 예를들어

여러분이 MPEG 미디어 파일을 전송하고자 하고, 또한 영상과 음성을 둘 다 전송하고자 한다면, 여러분은 영상 및 음성에 대하여 각각 RTP 세션을 만들고 각각의 세션을 통하여 오디오 및 비디오 데이터스트림을 분리하여 전송해야만 한다는 것이다.

Processor 를 통한 출력 데이터의 획득방법

일단 프로세서의 트랙에 대한 RTP 전송 포맷이 설정되고 프로세서가 Realize 되었다면, 해당 프로세서의 출력 DataSource 를 획득할 수 있다고 이미 언급하였다. 독자들은 getDataOutput 을 통하여 DataSource 의 형태로서 프로세서의 출력을 획득하며 이렇게 획득된 데이터 소스는 데이터 소스의 성질에 따라서 PushBufferDataSource 또는 PullBufferDataSource 로 구성되어진다. 프로세서의 출력 데이터 소스는 createSendStream 을 통하여 SessionManager 에 연결되며, 세션 메니져는 실제적인 전송을 하기에 앞서서 초기화가 되어야 한다.

만약 데이터소스에 여러 개의 SourceStream 이 존재한다면 어떻게 해야할까. 이때에는 각각의 Source Stream 은 각각 별개의 RTP 스트림으로 보내져서 동일한 세션이나 혹은 다른 세션을 통하여 전송을 할 수 있다. 가령 예를들어서 DataSource 가 오디오 및 비디오 스트림을 가지고 있다면, 각각의 독립된 RTP 세션이 오디오를 위한 세션과 비디오를위한 세션으로 분리되어 만들어져야 한다. 지난 강좌중에 데이터 복제 방법에 관하여 언급한적이 있다. 프로세서의 출력물은 이렇게 데이터 복제 방법을 통하여 다시 DataSource 형태의 결과물을 얻을 수 있으며, 이렇게 복제된 데이터들은 동일한 세션 혹은 별도의 세션을 통하여 전송에 참여할 수 있다.

Packet 지연에 대한 제어방법

패킷지연은 네트워크를 통해서 전송되어진 각각의 RTP 패킷에 의해 최종적으로 표현이 되어질때까지 소요되는 시간을 의미한다. 이 시간은 최소 종단간 지연시간을 결정하게된다. 패킷수가 많다면 헤더의 오버헤드는 적지만, 지연시간을 많이 요구하게 되며, 심지어는 전송중의 패킷 손실이 커지게된다. 예를들어 보면, 오디오 데이터의 경우 0 - 200 ms 사이에서 표현되는 패킷데이터를 얻어야만한다. 이러한 제약은 수신단의 적절한 버퍼크기 조정방법을 허용하고, 각각의 패킷화 코덱은 부호화에 이용되는 기본 패킷화 인터벌을 값을 가지게 된다. 만약 코덱이 이러한 인터벌의 변경을 허용한다면, 코덱은 PacketSizeControl 의 정보를 보내게되며, 패킷화 간격은 setPacketSize 메쏘드를 통하여 변경되어 질 수 있다.

비디오 스트림의 경우, 하나의 비디오 프레임은 여러 개의 RTP 패킷내에서 전송되어 지며 각각의 전송 패킷의 크기는 이용하는 네트워크의 최대 전송 단위 즉, MTU 에 의해 제약을 받게된다. 이 값은 또한 패킷화 코덱의 PacketSizeControl 의 setPacketSize 메쏘드를 통하여 새로운 값으로 설정될 수 있다.

DataSink 를 이용한 RTP 데이터 송신하기

먼저 [리스트 1]의 프로그램을 살펴보도록 하자. 이 프로그램에서는 먼저 원하는 오디오 포맷을 지정하고 그 포맷을 캡쳐장치에게 요청을 하여 캡쳐장치의 정보를 얻어온다. 이렇게 획득한 캡쳐장치 정보를 통하여 프로세서를 생성하고, 초기화 한다. 다음으로는 프로세서의 출력포맷을 RAW 포맷으로 지정하고, 프로세서에서 포맷 정보를 획득하여 오디오와 비디오의 트랙정보중 오디오 트랙정보만을 획득하고, 이를 GRM_RTP 포맷으로 변경하는 작업을 수행하여 트랙을 활성화 시킨다. 이후 프로세서를 Realize 시키고, 실제 통신을 위한 데이터 싱크를 생성하여 구동을 시킴으로서 통신이 수행되게 된다. 아래의 예제에서는 오디오 트랙부분만을 전송하기 위한 예제이지만, 반대로 오디오 트랙 검출 부분을 막고, 비디오 트랙부분만을 전송하도록 바꿀 수도있다. 앞서도 언급을 했지만, DataSink 를 이용할경우 데이터 소스에 존재하는 여러 개의 트랙중 첫번째 트랙만이 전송 가능하다는 점이다.

[리스트 1. DataSink 를 이용한 미디어 데이터의 전송]

```
AudioFormat format= new AudioFormat(AudioFormat.LINEAR, 8000, 8, 1);
Vector devices= CaptureDeviceManager.getDeviceList( format);
CaptureDeviceInfo di= null;
if (devices.size() > 0) {
    di = (CaptureDeviceInfo) devices.elementAt( 0);
}
else {
     System.exit(-1);
}
try {
        Processor p = Manager.createProcessor(di.getLocator());
} catch (IOException e) {
  System.exit(-1);
}catch (NoProcessorException e) {
  System.exit(-1);
```

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

if (encodingOk) {
   processor.realize();
   DataSource ds = null;
   try {
      ds = processor.getDataOutput();
   } catch (NotRealizedError e) {
      System.exit(-1);
   }
   try {
      String url= "rtp://224.144.251.104:49150/audio/1";
      MediaLocator m = new MediaLocator(url);
      DataSink d = Manager.createDataSink(ds, m);
      d.open();
      d.start();
   } catch (Exception e) {
      System.exit(-1);
```

```
   }
}
```

다음으로는 [리스트 2]를 살펴보도록 하자. 아래의 프로그램에서는 캡쳐장치를 찾고, 프로세서를 생성하는 부분까지는 동일하다. 이후의 작업에서 프로세서에서 이용가능한 트랙정보를 얻어내어 트랙이 비디오 트랙일 경우에 트랙정보를 세팅하고 활성화였고, 프로세서의 출력 데이터 소스를 얻는 부분이다.

[리스트 2. DataSink 를 이용한 비디오 데이터의 전송]

```
private void createProcessor() {
    FindCaptureDriver();
          try {
              processor = Manager.createProcessor(src);
          } catch (NoProcessorException npe) {
          } catch (IOException ioe) {
          }
          boolean result = waitForState(processor, Processor.Configured);
          TrackControl tracks[] = processor.getTrackControls();
          for (int i = 0; i < tracks.length; i++) {
            Format format = tracks[i].getFormat();
            if(tracks[i].isEnabled() && format instanceof VideoFormat) {
                    Dimension size = ((VideoFormat)format).getSize();
                    float frameRate = ((VideoFormat)format).getFrameRate();
                    VideoFormat videortpFormat = new
                        VideoFormat( VideoFormat.JPEG_RTP, size,
                             Format.NOT_SPECIFIED, Format.byteArray,
                             frameRate);
                    tracks[i].setFormat(videortpFormat);
            }
        }
  ContentDescriptor cd = new ContentDescriptor(ContentDescriptor.RAW);
  processor.setContentDescriptor(cd);
  result = waitForState(processor, Controller.Realized);
  dataOutput = processor.getDataOutput();
  }
```

앞의 프로그램을 살펴본 독자라면 아마도 좀 다른 점을 찾아냈을 것이다. 이전까지 Player 로 로컬에서의 프리젠테이션을 구현했었는데 Processor 를 이용한다면 어떻게 화면에 표시가 되는것일까. 이 문제를 해결하기 위해서 이전 강좌에서 구현을 하였던 데이터 소스의 복제가 필요하다. 다음의 [리스트 3] 을 살펴보자.

[리스트 3] 데이터 소스의 복제 과정

```
public void FindCaptureDriver(){
        Vector vectorDevices = CaptureDeviceManager.getDeviceList (null);
        CaptureDeviceInfo captureDeviceInfo;
        int nCount = vectorDevices.size();
        Format arrFormats[];
        for (int i=0; i<nCount; i++) {
            captureDeviceInfo =
CaptureDeviceInfo}vectorDevices.elementAt(i);
            arrFormats = captureDeviceInfo.getFormats();
            for (int j=0; j<arrFormats.length; j++) {
               if (arrFormats[j] instanceof VideoFormat) {
                    videoDeviceName = captureDeviceInfo.getName();
                    System.out.println(videoDeviceName);
                    break;
                }else if(arrFormats[j] instanceof AudioFormat) {
                }
            }
        }
        dataSource = JMFUtils.createCaptureDataSource(null,null,
                     videoDeviceName,null);
        src = Manager.createCloneableDataSource(dataSource);
        clone = new Clone(this);
        if(! clone.open(((SourceCloneable)src).createClone())){
            System.exit(0);
        }
 }

private void createTransmitter() {
          String rtpURL = "rtp://" + ipAddress + ":" + port + "/video";
```

```
            MediaLocator outputLocator = new MediaLocator(rtpURL);
            try {
              rtptransmitter = Manager.createDataSink(dataOutput,
                               outputLocator);
              rtptransmitter.open();
              rtptransmitter.start();
              dataOutput.start();
            } catch (MediaException me) {
                System.out.println("MediaException in createTransmitter" + me);
                System.exit(0);
            } catch (IOException ioe) {
                System.out.println("IOException in createTransmitter" + ioe);
                System.exit(0);
            }
}
```

리스트에서 보듯이 먼저 캡쳐장치를 찾고 그에 해당하는 데이터 소스를 얻어온다. 이렇게 획득한 데이터소스는 전송을 위해 이용되며, Processor 의 입력으로 들어가게 된다. 그러나 이러한 과정에는 로컬에서의 모니터링을 위한 프리젠테이션 기능이 없으므로, 이를 위해 데이터 복제과정을 통해 새로운 데이터소스를 하나 복제해내고, 이렇게 복제된 데이터소스는 전송이 아닌 로컬에서의 Player 의 입력 데이터 소스에 이용되게 된다. 데이터 소스를 이용한 전송 부분은 [리스트 1]에서 살펴본 바와 다른 점이 없다. 주의 할점은 DataSink 를 구동 시키면서 데이터 소스 역시 같이 구동을 시켜주어야 한다는 점이다.

세션 메니져를 이용한 RTP 데이터의 전송방법

세션 메니져를 이용하여 RTP 데이터를 전송하는 기본 과정은 다음과 같다. 첫번째로 Processor 를 생헝하고 RTP 에서 규정한 포맷에 맞도록 각 트랙 정보를 설정하는 과정이다. 두번째 과정은 생성된 프로세서로부터 출력되는 DataSource 를 획득하는 부분이며 , 세번째 과정은 이전에 생성되고 초기화된 Session Manager 의 createSendStream 을 호출하는 부분이며, 네번째 단계는 Session Manager 의 startSession 을 호출하여 전송을 시작하는 부분이며, 다섯번째 단계는 SendStream 을 이용하여 전송과정을 컨트롤하는 부분이며 마지막으로 이러한 컨트롤을 위하여 SendStreamListener 를 등록하여 SendStream 에 대하여 발생하는 이벤트얻는 부분으로 구성이 된다. 물론 위에서 언급한 모든 과정을 전부 다

구현할 필요는 없으며, 독자들의 요구에 부합되도록 선택하거나 추가하도록 한다. 먼저 Send Stream을 만드는 방법부터 알아보자.

Send Stream 생성하기

세션 메니져가 데이터 전송을 하기 이전에, 전송되고자 하는 데이터를 어디에서 얻어야하는가를 알아야 한다. 독자들이 새로운 SendStream 을 생성할 때 Session Manager 에게 데이터를 획득할곳으로부터의 데이터소스를 인자로 넘겨주어야 한다. 주의할 점은 데이터 소스는 여러 개의 데이터 스트림을 포함할 수 있으므로, 이용하고자 하는 세션에서 스트림의 인덱스 정보를 알려주어야 한다. 또한 여러분들은 createSendStream 에게 서로다른 데이터 소스를 전달하거나 혹은 서로다른 스트림 인덱스 정보를 넘겨줌으로서 여러 개의 다중 Send Stream 을 생성하는 것이 가능하다. 이러한 과정을 마친후에 세션 메니져는 Source 스트림의 포맷에 대한 확인 과정을 통하여 요청한 포맷이 등록된 PayLoad Type 인지를 확인하게 된다. 만약 데이터의 포맷이 RTP 포맷이 아니거나 또는 RTP 포맷에 지정되지 않는 payload type 이라면 해당 메시지와 더불어 UnSupportedFormatException 예외를 발생하게 된다. 또한 JMF 에서는 다이나믹 Payload 를 지원하여서 Session Manager 의 addFormat 메쏘드를 이용하여 RTP 포맷을 연결 시킬 수 있다.

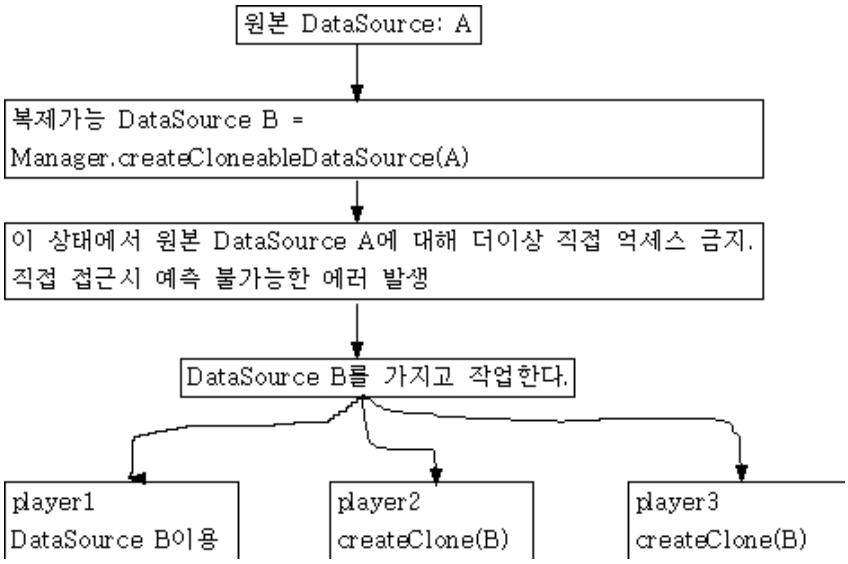
Cloneable 데이터 소스의 이용

대부분의 RTP 를 이용하는 애플리케이션 환경에서는 여러 개의 RTP 세션을 통하여 특정 스트림을 보내는 작업을 요구하거나 하나의 데이터 스트림을 다양한 포맷으로 변경을한 후에 이를 다시 여러 개의 RTP 세션으로 전송하는 작업을 요구하기도 한다. 이러한 환경하에서는 데이터가 캡쳐되어지는 Processor 로부터의 출력 데이터소스를 복제해야한다. 이러한 작업은 Manager 를 통하여 복제가능한 cloneable 데이터 소스를 생성하고 복제가능한 데이터소스에 대하여 getClone 를 호출함으로서 원하는 결과를 얻을 수 있다. 이렇게 복제된 각각의 데이터 소스에 대하여 새로운 Processor 가 생성되어 RTP 세션을 통한 전송이 이루어 질 수 있다. 그러나 이렇게 데이터를 복제하는 과정에서는 특히 주의를 해야하는 점이있다. [그림 1]을 살펴보자.

[그림 1]에서 나타낸 것 처럼 복제를 하고자 하는 원본 데이터 소스를 A 라고 하자. 먼저 Manager 의 createCloneableDataSource 를 통하여 원본 데이터 소스로부터 복제가능한 데이터소스 B 를 만들어야 한다. 이제 이렇게 생성된 복제가능한 데이터

소스인 B 를 통하여 createClone 을 이용하여 데이터 소스들을 생성할 수 있다. 그러나 복제가능한 데이터 소스인 B 를 생성하는 순간부터 원본 데이터 소스 A 는 더 이상 사용을 할 수없다. 실제로 원본 데이터소스에 억세스시에는 Exception 이 발생하게 된다.

[그림 1] 데이터 소스의 복제 과정



데이터 소스의 병합방법

만약 동일한 타입(예를들어서 오디오포맷등등..)의 여러 미디어 스트림들을 하나로 묶어 하나의 데이터소스를 나타내는 단일 스트림으로 만들고자 한다면, RTP Mixer 를 이용해야한다. 만약 병합 하고자 하는 스트림들이 서로 다른 데이터소스들로부터 생성이 된다면, 각각의 데이터소스로부터 MergingDataSource 를 생성할수있으며, 이것을 SessionManager 로 전달하여 스트림을 생성할 수 있다.

SendStream 동작 제어하기

스트림을 전송하는 것이 구현되더라도 실제적으로 이렇게 전송되는 스트림을 제어할 수 있는 방법이 없다면 아무런 소용이 없을것이다. JMF 에서는 RTPStream 의 start, stop 메쏘드를 이용하여 SendStream 의 동작을 제어할 수 있다. 즉, SendStream 을 start 시키는 것은 바로 네트워크를 통한 스트림의 전송을 의미하며, SendStream 을 stop 시키는 것은 스트림의 전송 중단을 의미한다. 또한 여러 개의 스트림의 전송을 위하여 각각의 SendStream 을 start 시켜야한다. 주의할 점은

SendStream의 stat , stop은 SendStream의 데이터 소스의 상태를 트리거링한다. 그러나, 만약 SendStream이 정지된 상태에서 데이터 소스가 독립적으로 구동된다면, 세션 메니져에 의하여 PushBuffer DataSource의 경우에는 데이터가 유실되거나 혹은 PullBuffer DataSource에서 데이터를 Pulling 할 수가 없게된다. 이 기간동안 어떠한 데이터도 네트워크를 통해 전달될수 없다.

단일 세션으로 스트림 데이터 전송하기

이제는 실제 코드를 예로 들어보자. [리스트 4]는 단일 RTP 세션을 통하여 캡쳐된 오디오 데이터를 전송하는 예제이다.

[리스트 4] 단일 RTP 세션을 통한 스트림의 전송

```
AudioFormat format = new AudioFormat(AudioFormat.ULAW,8000, 8, 1);
Vector devices= CaptureDeviceManager.getDeviceList( format);
CaptureDeviceInfo di= null;
if (devices.size() > 0) {
   di = (CaptureDeviceInfo) devices.elementAt( 0);
}
else {
   System.exit(-1);
}

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

processor.configure();
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] instanceof FormatControl) {
       if (((FormatControl)track[i]).
```

```
      setFormat( new AudioFormat(AudioFormat.ULAW_RTP,8000, 8,1)) ==  null) {
                  track[i].setEnabled(false);
       }
       else {
            encodingOk = true;
       }
    }
    else {
       track[i].setEnabled(false);
    }
}

processor.realize();
DataSource ds = null;
try {
    ds = processor.getDataOutput();
 } catch (NotRealizedError e){
    System.exit(-1);
 }

SessionManager rtpsm = new com.sun.media.rtp.RTPSessionMgr();
// rtpsm.initSession(...);
// rtpsm.startSession(...);
try {
     rtpsm.createSendStream(ds, 0);
} catch (IOException e)  {
     e.printStackTrace();
} catch( UnsupportedFormatException e) {
    e.printStackTrace();
}
```

먼저 캡쳐할 오디오 포맷을 지정해준후에 현재 시스템으로부터 우리가 지정한 오디오 포맷을 지원하는 캡쳐 디바이스가 있는지를 목록에서 검사한다. 이렇게 획득한 캡쳐 디바이스의 정보는 CaptureDeviceInfo 에 저장이 된다. 이제는 캡쳐정보의 getLocator 를 통하여 프로세서의 생성인자로 넘겨주어 프로세서를 생성하고, configure 작업을 수행하게된다. 이 과정까지 마친후에는 프로세서의

content Descriptor 를 설정해주며, 전송할 스트림의 트랙정보를 얻어와서 우리가 원하는 포맷으로의 변환을 수행하게된다. 이후 프로세서를 Realize 시키고, 프로세서의 출력 데이터 소스를 획득하여 이를 세션 메니져의 createSendStream 의 인자로 넘겨줌으로서 전송을 시작하게 된다. 세션 메니져의 생성과 초기화의 과정은 생략하였으므로 실제 프로그램에서는 구현해야 한다.

캡쳐된 오디오데이터의 멀티세션 전송방법

다음으로 살펴볼 [리스트 5]는 라이브 데이터로 입력된 오디오 데이터를 멀티 세션으로 전송기 위한 코드의 일부이다. 이 코드에서는 캡쳐된 오디오 데이터를 GSM 포맷으로 부호화를 한다. 먼저 프로그램을 살펴보자.

[리스트 5] 캡쳐된 오디오 데이터의 멀티세션 전송

```
AudioFormat format= new AudioFormat(AudioFormat.LINEAR,8000, 8,1);
Vector devices= CaptureDeviceManager.getDeviceList( format);
CaptureDeviceInfo di= null;
if (devices.size() > 0) {
    di = (CaptureDeviceInfo) devices.elementAt( 0);
}
else {
  System.exit(-1);
}

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

processor.configure();
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] instanceof FormatControl) {
     if (((FormatControl)track[i]).
          setFormat( new AudioFormat(AudioFormat.GSM_RTP,8000, 8,1)) == null)
          {
           track[i].setEnabled(false);
     }
     else {
         encodingOk = true;
     }
   }
   else {
     track[i].setEnabled(false);
   }
}

if (encodingOk) {
  processor.realize();
  DataSource origDataSource = null;
  try {
      origDataSource = processor.getDataOutput();
  } catch (NotRealizedError e) {
      System.exit(-1);
  }

DataSource cloneableDataSource = null;
DataSource clonedDataSource = null;

cloneableDataSource = Manager.createCloneableDataSource(origDataSource);
clonedDataSource = ((SourceCloneable)cloneableDataSource).createClone();
SessionManager rtpsm1 = new com.sun.media.rtp.RTPSessionMgr();

// 이 부분에서 세션 메니져가 초기화 되고 구동을 시작해야한다.
// rtpsm1.initSession(...);
// rtpsm1.startSession(...);
```

```
try {
 rtpsm1.createSendStream(cloneableDataSource, // Datasource 1
) catch (IOException e) {
      e.printStackTrace();
  } catch( UnsupportedFormatException e) {
      e.printStackTrace();
  }

try {
     cloneableDataSource.connect();
     cloneableDataSource.start();
  } catch (IOException e) {
    e.printStackTrace();
  }
 if (clonedDataSource != null) {
      SessionManager rtpsm2 = new com.sun.media.rtp.RTPSessionMgr();

      // 이 부분에서 두번째 세션 메니져가 초기화 되고 구동되어야 한다.
      // rtpsm2.initSession(...);
      // rtpsm2.startSession(...);

     try {
         rtpsm2.createSendStream(clonedDataSource,0);
     } catch (IOException e) {
        e.printStackTrace();
     } catch( UnsupportedFormatException e) {
        e.printStackTrace();
     }
  }
  }
  else {
       processor.deallocate();
       processor.close();
}
```

[리스트 5]는 다소 많은 분량의 코드이지만, 그 구조는 앞서 설명한 [리스트 4]와 유사하다. 우선적으로 해야하는 작업은 오디오 캡쳐 장치를 찾아서 8 비트 8kHz 의 캡쳐 포맷을 지원하는지를 조사하는것이다. 원하는 캡쳐 장치가 선택되면 이후의 작업은 캡쳐장치에 대한 프로세서를 생성하여 프로세서의 configure 를 수행하는 것이다. 프로세서가 congifured 된후에는 프로세서의 setContentDescriptor 를 통하여 포맷을 세팅하고, 프로세서에서 이용하는 데이터 소스의 각 트랙을 검사하여 이를 새로운 오디오 포맷인 GSM 으로 변경하는 작업을 수행한다. 트랙들의 데이터 포맷 변경이 종료되면 프로세서를 realize 시키고 프로세서의 출력 데이터 소스를 획득하여야 한다. 주의할 점은 이 부분에서 우리는 획득한 데이터 소스를 2 개의 RTP 세션으로 보내기를 원하기 때문에 프로세서의 출력 데이터소스를 복제하고 복제된 데이터 소스를 두번째 세션 매니져에게로 전달하는 과정을 거쳐야 한다. 먼저 첫번째 세션 메니져를 생성하고 첫번째 데이터 소스를 SendStream 생성을 위한 인자로 넘긴다. 두번째의 RTPSessionMgr 를 생성하고 복제된 데이터 소스를 인자로 넘겨줌으로서 다중 세션을 통한 데이터 전송이 이루어지게된다.
이제까지 단일 포맷에 대한 멀티세션 전송을 구현해보았으니 이번에는 멀티 포맷으로 구성되는 데이터 소스를 멀티 세션으로 전송하는 방법을 알아보자.
[리스트 6]의 코드를 먼저 살펴보자.

[리스트 6] 멀티 포맷오디오 데이터의 멀티세션 전송

```
AudioFormat format = new AudioFormat(AudioFormat.LINEAR, 8000, 8, 1);
Vector devices= CaptureDeviceManager.getDeviceList( format);
CaptureDeviceInfo di= null;

if (devices.size() > 0) {
    di = (CaptureDeviceInfo) devices.elementAt( 0);
}
else {
    System.exit(-1);
}

DataSource origDataSource= null;
try {
    origDataSource = Manager.createDataSource(di.getLocator());
} catch (IOException e) {
    System.exit(-1);
} catch (NoDataSourceException e) {
```

```
    System.exit(-1);
}

SourceStream streams[] = ((PushDataSource)origDataSource).getStreams();
DataSource cloneableDataSource = null;
DataSource clonedDataSource = null;
if (streams.length == 1) {
   cloneableDataSource = Manager.createCloneableDataSource(origDataSource);
   clonedDataSource = ((SourceCloneable)cloneableDataSource).createClone();
}
else {
}
Processor p1 = null;
try {
   p1 = Manager.createProcessor(cloneableDataSource);
} catch (IOException e) {
  System.exit(-1);
} catch (NoProcessorException e) {
  System.exit(-1);
}

p1.configure();
TrackControl track[] = p1.getTrackControls();
boolean encodingOk = false;
for (int i = 0; i < track.length; i++) {
   if (!encodingOk && track[i] instanceof FormatControl) {
      if (((FormatControl)track[i]).
        setFormat( new AudioFormat(AudioFormat.GSM_RTP,8000,8,1)) == null) {
                      track[i].setEnabled(false);
      }
      else {
          encodingOk = true;
      }
}
else {
  track[i].setEnabled(false);
```

```
}
}
if (encodingOk) {
     processor.realize();
     DataSource ds = null;

      try {
         ds = processor.getDataOutput();
      } catch (NotRealizedError e) {
          System.exit(-1);
      }
     SessionManager rtpsm1 = new com.sun.media.rtp.RTPSessionMgr();
     // rtpsm1.initSession(...);
     // rtpsm1.startSession(...);

     try {
      rtpsm1.createSendStream(ds, // first datasource
                              0); // first sourcestream of first datasource
     } catch (IOException e) {
                 e.printStackTrace();
     } catch( UnsupportedFormatException e) {
       e.printStackTrace();
     }
  }

 // 이 부분에서 위에서 수행한 프로세서 생성작업을 반복한다.
// 여기서는 복제된 데이터 소스를 이용하고 이 데이터 소스의 포맷을 DVI 로
//  변경한다. 즉, 복제된 데이터 소스를 이용하여 프로세서의 입력인자로
// 전달하여 프로세서를 생성하고  프로세서의 트랙정보를 DVI 로 바군후에
// 프로세서의 출력 데이터 소스를 획득하여 데이터를 전달해 준다.
```

[리스트 6]은 캡쳐된 오디오 데이터소스를 여러가지 포맷으로 부호화 하고 이를 멀티 세션으로 전송하는 방법을 구현한 것이다. 일단 구현상 입력되는 이터소스는 하나의 스트림으로 구성이 되어있다고 가정한다. 입력되는 데이터 소스는 일단 복제가 되고 이 복제된 데이터 소스를 통하여 두번째 프로세서가 만들어져야 한다.

이렇게 생성된 두번째 프로세서내의 트랙들은 각각 GSM 포맷과 DVI 포맷으로 설정되고 출력 데이터 소스들은 두개의 서로다른 RTP 세션 매니져에게로 전달이 된다. 이 프로그램에서는 트랙의 수가 2개 이상이면 첫번째 트랙은 GSM으로 세팅을 하고, 나머지들은 DVI 포맷으로 세팅을 하게된다. 이 프로그램에서는 동일한 데이터 소스가 두개의 독립된 RTP 세션 매니져에게로 전달이 되는데 하나는 인덱스 0번으로 그리고 두번째 스트림은 인덱스 번호 1로 전달되어 멀티 포맷을 통한 멀티 세션 전송을 구현하게 된다.

RTP Socket 을 이용한 RTP 스트림의 전송

이번에는 RTPSocket 을 이용하여 RTP 미디어 스트림을 전송하는 방법을 알아보자. 전송을 위한 RTPSocket을 이용하기 위해서는 MediaLocator 내에서 RTP의 변형인 "Ratibor" 프로토콜을 이용하여 RTP DataSink를 생성해야한다. Manaer는 <protocol package-prefix>.media.datasink.rtpraw.Handler 로 부터 DataSink를 생성하기 위한 시도를 하게된다. 세션 메니져는 네트워크를 통해 전송되어질 준비가된 각각의 RTP 데이터들을 준비하게되며 <protocol package-prefix>.media.protocol.rtpraw.DataSource 로부터 생성된 RTPSocket 으로 패킷 데이터들을 전달하게된다. <protocol-prefix>.media.protocol.rtpraw.DataSource 에서 생성된 RTPSocket이 RTPSocket에 대한 독자들의 구현부분이 된다.
주의할점은 JMF에서는 이러한 RTPSocket의 표준적인 구현방법을 정의하지 았다는 점이다. 실제로 RTPSocket의 구현은 RTPSocket의 이용한 독자들이 현재 이용하는 네트워크의 전송 프로토콜에 전적으로 전적으로 의존하게된다. 독자들이 구현하는 RTPSocket 클래스는 반드시 <protocol-prefix>.media.protocol.rtpraw.DataSource 위치에 있어야 하며, 이용하는 네트워크상에서의 RTP 패킷들의 전송에 대한 책임은 구현하는 사람들에게 있다는 점이다.

[리스트 7] RTPSocket 을 이용한 데이터의 전송구현

```
AudioFormat format = new AudioFormat(AudioFormat.LINEAR, 8000, 8,1);
Vector devices= CaptureDeviceManager.getDeviceList( format);
CaptureDeviceInfo di= null;
if (devices.size() > 0) {
   di = (CaptureDeviceInfo) devices.elementAt( 0);
else {
   System.exit(-1);
}
try {
```

```
  processor = Manager.createProcessor(di.getLocator());
} catch (IOException e) {
  System.exit(-1);
} catch (NoProcessorException e) {
  System.exit(-1);
}
processor.configure();
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] instanceof FormatControl) {
     if (((FormatControl)track[i]).
           setFormat( new AudioFormat(AudioFormat.GSM_RTP,
                                8000,8, 1)) == null) {
            track[i].setEnabled(false);
      }
      else {
         encodingOk = true;
      }
   }
   else {
     track[i].setEnabled(false);
   }
 }
if (encodingOk) {
     processor.realize();
     DataSource ds = null;
     try {
        ds = processor.getDataOutput();
     } catch (NotRealizedError e) {
        System.exit(-1);
     }

// 위의 데이터 소스를 RTP 데이터싱크를 생성하기 위위하여 manager 에게로
```

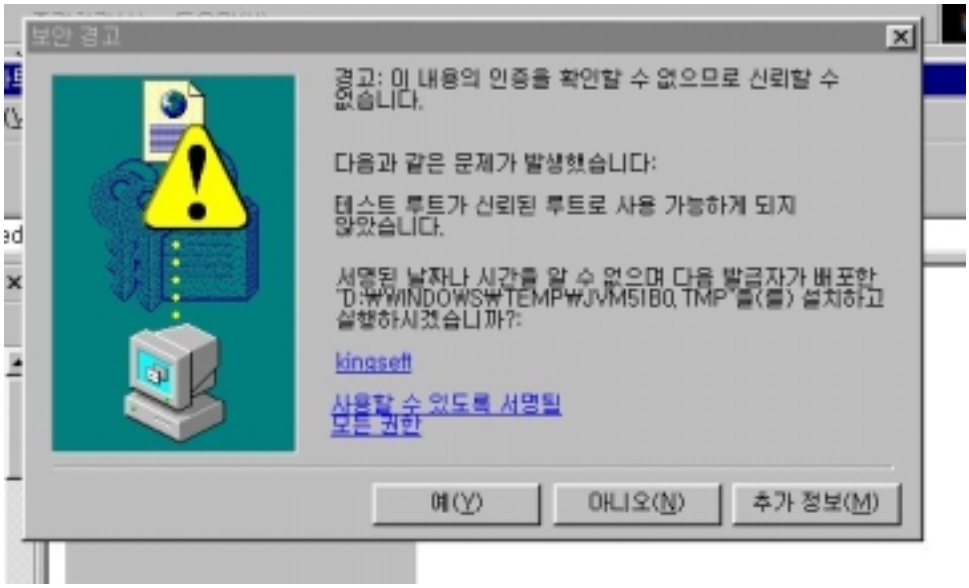
```
// 전달한다.
// 여기서 RTP 데이터싱크는 오디오 데이터를 멀티캐스팅하게된다.
try {
    MediaLocator m = new MediaLocator("rtpraw://");
    DataSink d = Manager.createDataSink(ds, m);
  d.open();
  d.start();
} catch (Exception e) {
   System.exit(-1);
}
}
```

주의할점은 try{ }catch 구분으로 이루어진 부분안에서 manager 는 <protocol.prefix>.media.protocol.rtpraw.DataSink 에서 데이터 싱크를 탐색하게 되며, 데이터 싱크는 <protocol.prefix>.media.protocol.rtpraw.DataSource 에서 RTPSocket 를 생성하게되며 모든 RTP 데이터를 이 소켓으로 전달하게된다.

IE 웹브라우져에서의 애플릿을 통한 클라이언트 컴퓨터의 자원접근문제

이제는 RTP 전송 프로그램을 애플릿으로 구현하여 IE 와 같은 웹브라우져에서 구동시킬때의 문제점에 대하여 알아보자. 먼저 [그림 1]부터 살펴보자.

[그림 1] IE 의 보안인증 윈도우



일반적으로 IE 와 같은 웹브라우져에서 애플릿을 이용하여 클라이언트 컴퓨터의 자원을 직접접근하는 캡쳐 작업 같은 경우에는 com.ms.securityException 이 발생되고 보안에 관한 예외처리가 발생하게된다. 이부분의 해결을 위하여 독자들은 IE 에서의 문제해결을 위해서 몇가지 단계를 거쳐 Signed Applet 을 생성해야한다. 먼저 [그림 1]부터 살펴보자. 아래의 그림은 Signed Applet 을 통하여 IE 상에서 인증서창이 뜨는 부분이다. 지금은 정식 인증서가 아닌 테스트 용도로 작성하였기에 실제 상업적인 인증서에서의 표시와는 다르게 나타난다. 아래와 가은 보안 경고 창이 생성되었을 경우 [예] 버튼을 누름으로서 클라이언트 컴퓨터의 로컬 자원에 억세스하는 기능, 즉 캡쳐장치에 접근하여 오디오 및 비디오를 캡쳐할 수 있게된다. 자. 그럼 실제적인 구현을 위하여 IE 에서의 Signed Applet 생성 방법에 관하여 알아보자.

먼저 Microsoft 싸이트의 자바 페이지(http://www.microsoft.com/java) 에서 Microsoft SDK for java 4.0 을 다운로드 하고 자신의 컴퓨터에 인스톨 시켜야한다. 또한 자신의 시스템에 가장 최신 버전의 Java VM 이 설치되었는지 확인하고 그렇지 않은경우 역시 이곳에서 JVM 을 다운로드 받아 설치하기 바란다. 설치를 마친후에는 Microsoft SDK for java 의 bin 디렉토리를 여러분이 설정해두었던 path 에 추가를 해주어야 한다. PATH=%PATH%;C:₩Microsoft SDK for Java4.0₩bin; 과 같이 기존의 Path 에 추가를 해주면 된다. 이와 같은 모든 설치과정을 마치면 독자들은 makecert , cabarc, cer2spc, signcode 등과 같은 컴맨드라인상의 프로그램들을 이용할 수 있다.

makecert 이용하기

IE 에서 Signed Applet 을 이용하기 위해 가장 먼저 해야할 작업은 도스모드 에서 makecert 를 이용하여 인증서를 만드는 부분이다. makecert 의 사용방법은 다음과 같다.

makecert -sk [인증서 별명] -n " CN=[인증서이름]" [인증서파일]

다음으로는 makecert 에 대한 몇가지 옵션을 살펴보기로 한다.

[makecert 명령어의 옵션]

-sk " KeyName"

레지스트리에 보관되어있는 key 의 이름으로 저장되어있는 key 의 이름이 없는 경우에는 지정한 keyName 으로 레지스트리에 새롭게 keyName 을 생성한다.

-ss “ Store”
인증서를 실제적으로 보관할 장소에 대한 이름을 지정할 수 있다.

-sr “ Location”
레지스트리 내부에 인증서를 보관할 직접적인 위치를 지정한다.
레지스트리의 키값중 CurrentUser, LocalMachine 에 저장을 한다.

-# “ Number”
1-130 사이의 고유 번호를 부여할수 있다.

-$ “ Authority”
인증서를 발급한 기관의 형태를 나타낸다(개인 | 공식기관) 으로 설정된다.

-n “ X.509 name”
X.509 의 구분으로 지정되는 이름을 명시한다. (예 : CN = [인증이름])

-?
기본 옵션의 각 목록에 대한 설명을 나타낸다.

-!
확장 옵션의 각 목록에 대한 설명을 나타낸다.

위와 같은 옵션들을 통하여 makecert 과정을 수행하면 Public key 와 Private key 의 쌍이 생성이 되고, Private key 는 Registry 에 저장이 되게된다. 또한 인증서가 제시한 인증서파일명으로 생성되어진다. 아래와 같이 예를 살펴보자.
makecert -sk kingseftCert -n "CN=kingseft" kingseft.cer

이 작업을 통하여 인증서의 별명은 kingseftCert 가 되며, 실제 인증서의 이름은 kingseft 가 되며, 생성되는 인증서 파일은 kingseft.cer 파일이 된다

cert2spc
앞단계의 makecert 프로그램으로 원하는 이름으로 인증서를 생성하였다면, 다음 단계는 바로 .spc 형식의 인증서를 생성해야한다. Java 코드에서는 sign 을 할경우 .spc 형태의 서명파일을 필요로 하기 때문에 앞서 만든 cer 파일을 .spc

파일로 변환시키는 cert2spc 프로그램을 수행시켜야 한다. 사용방법은 다음과 같다.
cert2spc [인증서 파일명] [.spc 파일]
ex> cert2spc kingseft.cer kingseft.spc

이 작업을 통하여 앞서 생성했던 인증서인 kingseft.cer 파일은 java 코드로
sign 이 가능한 .spc 파일인 kingseft.spc 파일로 변환되어진다.

cabarc
다음 단계는 우리가 만들 애플릿 혹은 자바 파일들에 대한 서명작업을 부여하는
과정이다. 서명이 필요한 파일들은 cabarc 명령을 사용하여 cab 형식의 파일들로
만들어진다. 사용방법은 다음과 같다.
cabarc - r - p n [.cab 파일] [서명될 파일들]
ex> cabarc -r -p n kingseft.cab MyTestProgram.class

[cabarc 명령어의 옵션]
- L
cab 파일의 리스트 목록 보기 (cabarc L kingseft.cab)

- N
새로운 cab 파일의 생성 (cabarc N kingseft.cab .java)

- X
cab 파일로부터 특정 파일 추출하기 (cabarc X kingseft.cab My.java)

- c
작업할 파일 확인하기

- o
파일을 추출할 때 사용자에게 묻지 않고 바로 덮어쓰기

- m
압축된 형태를 지정하기 (LZX:<15.21>| MSZIP | NONE) 기본은 MSZIP

- p
파일명에 상대 경로를 기록하기

- P
파일 포함시에 지정된 접두어를 생략하기 (파일 경로의 일부가 기록됨)

- r
하위 디렉토리의 파일까지 모두 포함하기 (-p 옵션과 함께 사용된다.)

- S
cab 파일 내부에서 sing 정보를 둘 여유 공간을 확보할 때 이용

- I
cab 파일 생성시에 " cabinet set id " 를 지정하게 해준다.

위의 예제를 통하여 kingseft.cab 파일이 생성되고 MyTestProgram.class 는 서명된 파일로 인식된다. 주의할점은 현재 만들어진 cab 파일에는 아직까지는 인증서가 포함되어 있지 않고, 비밀키를 통한 서명작업이 이루어져 있지 않다는 점이다.

signcode
이제 우리가 해야 할 작업은 signcode 를 이용하여 앞서 만든 인증서를 cab 파일에 포함시키고, 레지스트리에 등록된 private key 로 서명작업을 해주는 과정이다.
사용 방법은 아래와 같다.
signcode -j javasign.dll -jp [레벨] -spc [.spc 파일] [.cab 파일]
ex> signcode -j javasign.dll -jp low -spc kingseft.spc -k kingseftCert
kingseft.cab

[signcode 명령어의 옵션]
-spc " file"
spc 를 포함하는 파일명을 명시한다.

-v " pvkFile"
Private Key 를 포함하고 있는 파일명을 나타낸다.

-k " KeyName"
레지스트리에 저장되어 있는 key 이름을 나타낸다.

-n “ name”
서명 작업을 할 내요에 대한 텍스트 이름을 나타낸다.

-l “ lnfo”
서명 작업을 할 내용에 대한 부가적인 URL 정보등을 나타낸다.

-p “ provider”
시스템 내의 암호화 시스템 제공자의 이름을 나타낸다.

-y “ type”
시스템 내의 암호화 시스템 제공자의 형태를 나타낸다.

-ky “ keyType”
인증키의 종류 (signature | exchange | 정수) 를 나타낸다.

-$ “ authority”
인증서를 공식적으로 인증한 기관의 종류 (개인 | 상업적단체) 중 하나를 나타낸다.

-a “ algorithm”
서명에 이용된 알고리즘으로 (md5 | sha1) 등을 이용한것임을 나타낸다.

-t “ URL”
타임 스탬프를 찍어줄 서버의 URL 정보를 나타낸다.

-tw “ number”
타임 스탬프의 시간간격을 초단위로 나타내준다.

-j “ dllName”
서명 작업에 필요한 부가적인 특성을 포함하는 DLL 파일등을 명시해준다.

-jp “ param”
DLL 파일로 넘겨줄 인자들을 명시해준다.

-c “ file”
부호화된 SPC 를 포함한 X.509 파일을 나타낸다.

-s “ Store”
인증서를 보관하는 인증서의 보관소명으로 기본은 myStore 이다.

-r “ location”
레지스트리 내의 인증서의 보관소 위치로서 localMachine, currentUser 값을
갖는다.

-sp “ policy”
서명된 인증서의 검증에 필요한 모든 인증서를 포함할것인가 아니면 SPC 보관소에
들어있는 인증서가 나올때까지 포함할것인가에 대한 정책으로서 chain , spcstore
값중 하나를 가지게된다.

-cn “ name”
인증서의 별명 이름을 나타낸다.

-x
서명작업을 하지 않고 타임스탬프 값만을 받을 것을 명시한다.

앞서서 많은 명령어의 옵션들을 설명하였는데, 실제로 독자들이 구현을 할때에는
대부분 아래의 순서를 따르면 된다. 최종 결과는 kingseft.cab 파일로
생성되어진다.

```
makecert -sk kingseftCert -n "CN=kingseft" kingseft.cer
cert2spc kingseft.cer kingseft.spc
cabarc -r -p n kingseft.cab MyProgram.class
signcode -j javasign.dll -jp low -spc kingseft.spc -k kingseftCert
kingseft.cab
```

이제 남은 작업은 앞서 생성된 .cab 파일을 html 파일에서 인식을 시키는 과정이다.
이 과정은 <PARAM>의 name 과 value 값에 의해 지정이된다. 사용법은 다음과 같다.

```
<Applet code="MediaAppletPlayer2.class" width=320 height=240>
<param name="cabbase" value="kingseft.cab">
</param>
</applet>
```

이와 같은 작업을 통하여 클라이언트측에서 애플릿을 다운로드 받게될 때 연결된 클라이언트측에 전자인증에 관한 창이 뜨며 서명된 애플릿을 사용할것인가를 묻게된다. 이때 [YES] 를 선택하면 해당 권한을 사용할 수 있게된다. 만약 독자들이 특정한 권한만을 부여하거나 혹은 모든 권한을 부여하는 작업을 하기 위해서는 프로그램 내부에서 몇가지 요구되는 작업이 있다. 먼저 Microsoft SDK Java 4.0의 security 패키지를 호출한다. 프로그램에 import com.ms.security.*; 를 추가하여야 하며, 보안사항의 설정을 위하여

```
try
{
    if (Class.forName("com.ms.security.PolicyEngine")!=null)
    {
        PolicyEngine.assertPermission(PermissionID.SYSTEM);
    }
}
catch (Throwable cnfe)  {
    System.out.println("인증부분 에러발생!!");
}
```

[부여 가능 권한의 종류]

1. SYSTEM
2. FILEIO
3. NETIO
4. THREAD
5. PROPERTY
6. EXEC
7. REFLECTION
8. PRINTING
9. SECURITY
10. REGISTRY
11. CLIENTSTORE
12. UI
13. SYSSTREAMS
14. USERFILEIO
15. MULTIMEDIA

위와 같은 부분이 애플릿 프로그램의 init() 코드의 처음에 추가되어야 한다. 위의 코드 부분에서 PolicyEngine.assertPermission(PermissionID.SYSTEM); 부분을 수정하여 독자들이 원하는 Security 관련 권한을 부여할 수 있다. 이제까지의 작업을 통하여 클라이언트측의 자원에 접근하여 애플릿을 통한 미디어 데이터의 캡쳐 작업과 전송 작업을 수행할 수 있다.

이번 강좌를 통하여 RTP 를 이용한 멀티미디어 스트림 데이터를 전송하는 실제 코딩방법을 논의하였으며, 단일 데이터 소스의 다중 송신과 포맷변환 송신에 관하여도 살펴보았다. 또한 DataSink 를 이용한 전송방법과 Session Manager 를 이용한 전송 방법등에 관하여 논의를 하였으며, 마지막으로는 서명한 애플릿을 이용하여 클라이언트 시스템의 자원을 억세스하고 데이터 송신기능을 수행하는 방법에 관해서도 논의하였다. 다음 연재에서는 RTP 송신에서 누락된 부분과 현재 베타 버전에서의 RTP API 의 변경부분에 대하여 언급하고 RTP 수신부분에 대한 논의를 하고자 한다.