제 1 강좌 : Java의 멀티미디어 전략 - JMF의 세계로

[서론]

JMF(Java Media Framework) 라는 것이 무엇일까요? 본 강좌에서는 Java의 확장 API인 JMF에 관한 자세한 소개와 더불어 인터넷 프로그래밍에서 가장 각광을 받고 있는 Sun의 Java진영에서 새롭게 발표한 확장 라이브러리 JMF를 사용해 화상채팅 시스템을 만들어 보는 기회를 갖고자 합니다. 또한 본 강좌를 통해 JMF를 이용하여 요즈음 인터넷 매니아라면 누구나 관심을 가질만한 주제인 화상채팅에 필요한 멀티미디어 파일 플레이어와 네트워크를 통한 송수신을 통해 진정한 JMF의 기능을 살펴봅니다. 자. JMF를 이용하기 위해 먼저 JMF의 종합적인 모습과 내부 구조를 살펴보아야 하겠지요. 그런다음 실제 JMF의 설치 및 구성과 발표된 패키지들, 그리고 프로그래밍을 하지 않더라도 지금 바로 설치해서 이용할 수 있는 멋진 기능들을 살펴보는 것으로 이번 강좌를 시작합니다.

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

[도입]

여러분들은 Sun의 Java 싸이트를 자주 방문하시나요? 진정한 Java 개발자라면 거의 매일같이 Sun의 Java 싸이트를 방문하겠지요? 최근에 Sun에서는 Java One이라는 행사를 통해서 Java의 현재 및 향후 기술에 대한 소개와 주요 관심 기술들 및 응용 분야에 대한 발표를 하였습니다. 발표되는 모든 기술들이 가장 최신의 기술이라고 말할 수 없지만, 개발자의 입장에서 기다렸던 기능들이 하나 둘 씩 발표되면서 Java 개발자로서의 개발 의욕을 높이고 있는 것이 사실입니다.

그러한 수많은 발표 기술중에 본 연재는 Java의 Java Media API로 발표되는 기술중에 가장 응용분야가 넓고 향후 발전가능성이 높은 JMF(Java Media Framework)에 대해 그 내부를 철저히 살펴보고 적용 가능한 응용분야에 대해 알아보고자 합니다. 또한 궁극적으로는 JMF를 다루면서 실제 요즈음 인기를 끄는 화상채팅 시스템에 대해 그 기본구조와 원리를 구현하는 것이 본 강좌의 최종 목표입니다. 그럼 먼저 JMF는 무엇이고, 어떠한 배경에서 탄생했으며 JMF의 내부구조와 구동원리는 어떻게되는지 살펴볼까요.

Java Media API의 소개

JMF 에 대해 소개하기전에 먼저 Java 에서 멀티미디어 데이터를 다루기 위해 발표된 모든 기술들에 간략히 살펴보기로 하죠. Java 에서는 멀티미디어 제어를 위해 Java Media API 라는 이름으로 패키지들을 발표했습니다. Java Media API 는 크게 8 개의 세부적인 API 들로 분류할 수 있습니다. 세부적인 API 의 목록은 다음과 같습니다.

Java Media API 의 종류 및 이용분야

1. Java 2D
- 2D 그래픽과 Image 의 조작
- Graphics 2D 에서 확장된 Graphics 이용부분

2. Java Media Framework
- 1.0 API 에서는 단순히 동기화된 미디어 데이터에 대한 playback 기능만 제공
- 2.0 API 에서 오디오 및 비디오데이터에 대한 capture, streaming 기능 제공

3. Java 3D
- 객체(object) 기반의 3D 그래픽 runtime 제공
- Java 기술로 제작된 VRML 97 reference browser

4. Java Sound
- software 로 구현된 sound processor 와 MIDI synthesizer 제공
- sound engine(java 2 SDK 1.2)과 sound API(beginning in java 2 SDK 1.3) 제공

5. java Speech
- 음성의 인식과 합성 기능 제공

6. Java Advanced Imaging
- 확장된 2D Image processing 기능
- java 2D API Interfaces 들의 구현

7. Java Telephony
- computer 와 전화 단말기의 통합
- javaphone API 의 하위 API 로서 향후 발표될 J2ME 기반 Mobile information device profile 제공

8. Java Shared Data Toolkit
- 애플리케이션 기반 Java 기술로서 사용자간 협동 작업을 가능하게하는 개발Tool 제공
- Session, URL, Registry를 통한 객체(object) 데이터의 공유기능

이와 같이 많은 기술들이 발표되었습니다. 본 강좌에서는 위와 같은 Java의 Media 처리 기술중에서도 JMF(Java Media Framework)를 집중적으로 다루어 보겠습니다. 자. 그럼 JMF는 과연 어떻게 개발되었고, 현재 어떻게 구성되어있는지 살펴보기로 하지요.

왜 JMF를 이용해야 하는가?

이러한 물음에 어떻게 답해야 할까요. 먼저 JDK(Java Development Kit)를 살펴보도록 하죠. 이제껏 발표된 JDK는 주로 디지털 오디오를 주된 목적으로 생각하였지만, 디지털 비디오나 심지어 MIDI(Musical Instrument Digital Interface)에 대한 중요성을 인식하지 못하였지요 또한 이전에 발표된 오디오 관련 API역시 단순한 디지털 오디오의 재생기능(playback)만을 수행하고, 어떠한 포맷으로도 저장하는 기능을 제공하지 못하였습니다. 이러한 상황에서 Sun의 Java 진영에서는 당시의 멀티미디어 관련 API들이 불합리하다는 생각을 하게되었고, 결국 객체지향, 풍부한 기능, 구조적인 멀티미디어 포맷을 지원하며, 단순한 오디오 데이터의 재생기능뿐만이 아니라 현재 이용되는 거의 모든 종류의 멀티미디어 데이터를 처리하기 위한 기능을 수행하는 API를 만들게 되었습니다. 그러한 구조가 바로 Sun의 Java Media API이며 , 대부분의 주된 관심이 Java Media API의 하부구조인 JMF의 탄생으로 이어지게 되었습니다.

SUN의 JMF 발표 단계

SUN과 Intel 그리고 Silicon Graphics 사는 다양한 환경의 플랫폼에서 구동가능한 멀티미디어 애플리케이션을 개발하려는 목적하에 JMF를 개발하게 되었습니다. Intel은 Win32 플랫폼으로, Sun은 Solaris 플랫폼으로, Slicon Graphics는 IRIX기반으로 서로 JMF를 개발하기로 하였습니다. 초기의 JMF API는 1997년에 Intel의 Win32 기반 JMF의 발표를 계기로 Sun과 Silicon Graphics가 각각 독자적인 API 발표를 하였습니다. 1998년 최종적으로 통합된 JMF가 SUN에의해 발표가 되었습니다. 그러나 최종적인 합의를 남겨두고 Silicon Graphics와 Intel이 향후 버전의 개발에서 불참하게 되었고, 후에 IBM이 Java에 큰 관심을 가지면서 계속해서 새로운 버전의 JMF가 개발되게 되었고 현재는 JMF 2.1 버전까지 발표되게 되었습니다.그럼 JMF 발표에 따른 변화된 모습을 간략하게 살펴보도록 하죠.

JMF 1.0으로부터 최근까지 이용되던 JMF 2.0까지의 변화된 모습입니다.

JMF 2.0 에서 새롭게 추가된 부분

- MPEG layer III (mp3) 오디오 부호화부분(encoder).
- Solaris상에서의 Cinepak pro video encoder .

- Processor로부터의 MonitorControl를 이용한 Audio and video monitoring.
- Microsoft의 Video for windows를 위한 Video monitoring.
- Non-realtime software video scaler.
- MimeManager를 통한 새로운 mimetype에 대한 등록기능(in the com.sun package).
- 완전하게 지원되는 FramePositioningControl.
- 쉬운 preview 접근 방식지원:
    - 송신과 수신을 위한 RTP MPEG audio 와 video 기능지원.
    - RTSP 수신기능 지원.
- JMStudio를 위한 향상된 GUI 디자인부분.

얼마전에 발표된 JMF 2.1 FCS 에서 새롭게 추가된 기능입니다.

- Linux를 위한 Performance Pack 추가.
- 완전한 RTSP client 부분지원.
- Solaris상에서의 SunRay를 위한 최적화된 video rendering 기능지원.
- Video scaling.
- bug 수정등등.

이상이 현재까지 발표된 JMF 2.1의 기능입니다. 간략하게 정리하면 JMF 2.1 API는 이전에 발표된 JMF 1.0 이후 버전들에 대한 가장 최근의 확장판이며 JMF 2.1을 통해서 미디어 데이터에 대한 캡쳐기능과 저장기능이 가능하고, , 미디어 데이터의 재생 기능중에 수행되는 processing type을 제어할수 있습니다. 또한 미디어 데이터 스트림에 대한 custom processing도 지원을 하며, JMF 2.1 에서는 plug-in API를 지원하여 개발자들이 JMF Functionality를 보다 쉽게 최적화하는 기능을 제공합니다.
무엇보다도 여러분은 JMF를 통해서 음성과 영상을 캡쳐하고, 캡쳐된 데이터를 컴퓨터의 저장장치에 기록하고, 상대방의 컴퓨터에 네트워크를 통해서 전송이 가능하다는 점에 가장 큰 관심을 두실것이라고 생각됩니다. 자. 그럼 현재발표된 최신의 JMF 2.1은 어떠한 형식의 패키지로 발표되었는지 살펴보도록 하죠.

JMF 2.1의 패키지 종류들
JMF2.1을 이용하기 위해서 발표된 JMF 패키지는 총 3가지로 분류됩니다.

- Java™ Media Framework 2.1
  클라이언트를 위하여 사용하는 운영체재의 종류에 상관없이 사용가능한 크로스 플랫폼 버전입니다.

전적으로 Java 로 작성되어있으며, Java Compatible clients를 위해 사용됩니다.
또한 이 버전에서는 JMFCustomizer라는 패키지를 제공하여 배포가능한 JAR 파일을 만드는 방법을 제공해 줍니다.
Java 클라이언트에게 JMF를 설치하기 위해서 이 패키지를 설치해야 합니다. 이 패키지에는 Java 클라이언트 환경을 위하여 서명된 JMF 압축파일인 jmf.jar 파일을 포함하고 있습니다.

- Java Media Framework 2.1 with Solaris™ Performance Pack
  Solaris 플랫폼 전용으로 성능과 기능을 향상시킨 JMF 버전입니다.

- Java Media Framework 2.1 with Windows Performance Pack
  Windows 계열의 플랫폼 전용으로 성능과 기능을 향상시킨 JMF 버전입니다.

위에서 살펴본 바와 같이 3가지의 패키지가 제공되지만, 실제로 여러분의 운영체제나 사용목적에 맞추어 한가지를 선택하여 설치하시면 됩니다. 자. 그럼 JMF를 설치하기 위해 현재 자신이 사용하고 있는 컴퓨터 시스템의 사양을 확인해 보도록 하죠. 알맞은 하드웨어 사양 역시 위에서 언급한 JMF의 3가지 패키지에 따라서 조금씩 다릅니다.

JMF 설치를 위한 사양

1. Cross Platform Version JMF 설치시 요구되는 하드웨어 사양
하드웨어 요구조건

- 200 MHz Pentium, 160 MHz PowerPC, or 166 MHz UltraSparc
- 최소 64 MB RAM 필요

소프트웨어 요구조건

- JDK 1.1.6 이상.
- JMFCustomizer 구동을 위하여 Java 2 또는 JDK 1.x 과 Swing 1.1
- 선택사양: HotJava™ Browser 1.1.x.
- 선택사양: Netscape Communicator.

2. 윈도우 버전의 JMF 설치시의 사양
선택사양: Microsoft Internet Explorer 4.01 이상.
하드웨어 요구조건

- Pentium 100MHz 이상 추천.
- 빠른 디스플레이 속도를 위해서 16-bit VGA display mode 추천됨.
- 선택사양: SoundBlaster 호완되는 sound card.

소프트웨어 요구조건

- Windows 95/98 또는 Windows NT 4.0.
- JDK 1.1.6 이상.
- MPEG파일의 재생을 위해서 Microsoft DirectShow.
- JMF classes 과 native libraries (JMF 2.1에 포함).
- 선택사양: Microsoft DirectX 2.1 or greater.
- 선택사양: HotJava Browser 1.1.x (from Sun).
- 선택사양: Netscape Communicator.
- 선택사양: Microsoft Internet Explorer (Java Plug-in 필요) .

데이터 캡쳐 기능을 위한 요구사양

아래의 캡쳐 장치가 Windows Performance Pack에서 JMF 2.1에서 지원됩니다.:

- Video For Windows (VFW).
- DirectShow (DS),.

3. Solaris 플랫폼을 위한 요구사양
하드웨어 요구사양

- SparcStation 10, SparcStation 20, 또는 UltraSparc 기반의 workstation.
- 24-bit 프레임 버퍼 추천.

소프트웨어 요구사양

- Solaris 2.5.1 이상.
- JDK 1.1.6 이상.
- 선택사양: HotJava™ Browser 1.1.x.

- 선택사양: Netscape Communicator.

캡쳐 기능을 위한 요구사양

솔라리스의 Performance Pack에서 구동되는 JMF 2.1엣 ㅓ지원되는 캡쳐 디바이스 입니다.

- SunVideo 또는 SunVideoPlus.
- JavaSound.

이상으로 JMF 를 사용하기 위한 소프트웨어 및 하드웨어적인 요구조건을 전부 살펴보았습니다. 이 부분에서 주의하실 점은 여러분이 윈도우용의 JMF 를 이용하신다면 JMF 설치전에 반드시 DirecX 가 설치되어 있어야 한다는 점입니다. 또한 오디오와 비디오의 캡쳐를 위해서 JMF 설치전에 하드웨어 장치와 관련된 디바이스 파일들이 알맞게 설치되었는지 확인하셔야 합니다.

JMF 2.1의 설치 및 구동 방법

자. 이제 설치할 JMF 종류를 결정하셨나요? 결정을 하셨다면 이제는 실제로 SUN 싸이트에서 JMF 를 다운로드 받고 자신의 컴퓨터에 설치를 해야 합니다. 설치 방법역시 사용하는 플랫폼에 따라 약간의 차이는 있지만, 크게 다르지 않기 때문에 본 강좌에서는 윈도우 플랫폼을 기본으로 설치 방법과 실제 구동 방법에 대해 설명하겠습니다.

윈도우 환경에서의 JMF 2.1 설치와 구동 방법

JMF 의 윈도우 Performance Pack 는 JMF players 를 구동시키기 위하여 관련된 클래스 파일과 Native libraries 를 포함하고 있으며 윈도우에 최적화된 패키지 입니다. 자. 그러면 먼저 JMF 를 인스톨한후의 세팅 방법에 대해 알아보도록 하죠.

JMF 의 세팅 방법

1. 먼저 JMF를 설치할 대상 컴퓨터의 하드웨어 및 소프트웨어 사양을 확인합니다.
2. 만약 이전버젼의 JMF가 설치되어있다면 모두 제거해야 합니다.
3. 윈도우 버전의 JMF 설치파일을 구동시키면 InstallShield가 구동되며 설치를 시작합니다.
4. CLASSPATH와 PATH를 반드시 확인해 주시기 바랍니다.실제로 프로그램을 설치하는 과정에서 InstallShield 가 자동적으로 CLASSPATH와 PATH를 설정해줍니다. 만약 설

치후에 JMF의 올바른 설치 확인을 위해 다음과 같은 내용을 확인해야 합니다.

```
set
CLASSPATH=%WINDIR%₩java₩classes₩jmf.jar;%WINDIR%₩java₩classes₩s
ound.jar;.;%CLASSPATH%
```

또한 PATH 에서도 아래와 같은 사항을 확인해야 합니다.

```
set PATH=%WINDIR%₩System32;%PATH%        (on Windows NT)
set PATH=%WINDIR%₩System;%PATH%      (on Windows 95/98)
```

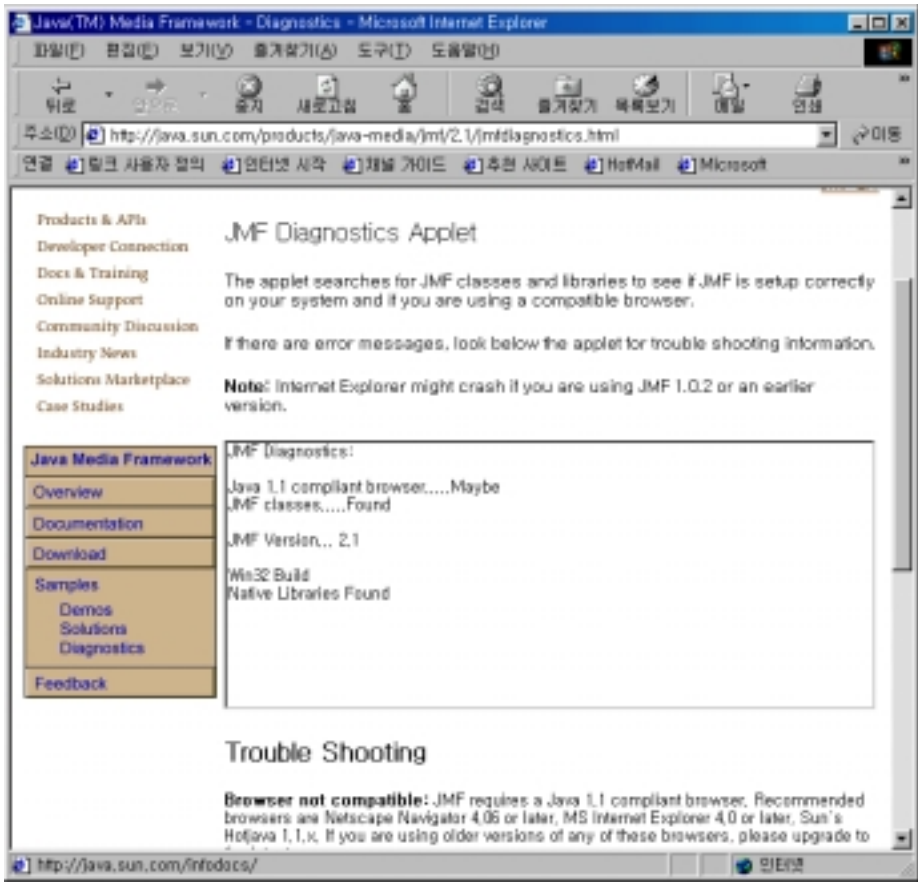
올바른 JMF 세팅 확인방법

위와 같은 과정을 통하여 jmf를 모두 설치한 이후에는 SUN 의 JMF 싸이트를 방문하셔서 JMF Diagnostics 애플릿 프로그램을 구동시켜 보셔야 합니다. 실제로 JMF 설치가 모두 올바르게 되었는지를 판별해주는 프로그램입니다. 테스트 해볼 프로그램이 있는 http://java.sun.com/products/java-media/jmf/2.1/jmfdiagnostics.html 주소로 방문하셔서 여러분의 인터넷 브라우져역시 JMF를 알맞게 인식하고 있는지 확인해야 합니다.

위의 싸이트를 방문하시면, JMF Diagnostics Applet이 구동되며, 현재 여러분의 시스템에 설치된 JMF 와 버전, 그리고 Native Libraries를 검사하게 됩니다. 검사를 마치면 아래와 비슷한 검사 결과를 표시하게 됩니다.

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

만약 위에서와는 다르게 에러 메시지 혹은 관련 라이브러리 파일을 찾을 수 없다고 메시지가 나오는 경우 상황에 맞게 다시 JMF를 설치하거나 적절한 조취를 취해 주어야 합니다.



[그림 1] JMF Diagnostics Applet 의 실행 결과

JMF Diagnostics Applet 의 문제 해결 방법

1. Browser not compatible:
JMF 는 자바 1.1 호환 브라우져를 요구합니다. 추천되는 웹 브라우져는 Netscape Navigator 4.06 or later, MS Internet Explorer 4.0 or later, Sun's Hotjava 1.1.x. 입니다. 만약 이전 버젼의 브라우져를 사용한다면 최신 버젼의 브라우져로 다시 설치하셔야 합니다. 또하나 주의해야 할 점은 실제 이전 버전의 브라우져를 사용한다면, 최신 버젼의 브라우져를 설치하구요. 그런 다음에 다시 JMF 를 재 설치 해야 합니다. 이와 같은 과정을 통해서만 JMF 를 설치하는 과정에서 Netscape 를 인식해서 필요한 파일들도 다시 정리해 줍니다.

2. Classes not found:
JMF classes 및 CLASSPATH 환경 변수에 제대로 설정이 되어있지 않을경우 발생합니다. CLASSPATH 에서 jmf.jar 를 기록해주었는지 확인해야 합니다. 일반적으로

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

3. Libraries not found:

JMF를 설치후에 생성되는 JMStudio 프로그램이 캡쳐가 안된다고 할때.. 가장 흔하게 나타나는 문제점이 바로 이 부분입니다. 이 부분에서 에러가 발생하면 캡쳐 프로그램을 수행하는 경우 예측못할 에러가 발생합니다. 만약 윈도우나 솔라시스같은 환경에서 캡쳐 관련 프로그램을 작성한다면 JMF 자체도 역시 Native Version의 JMF를 다운 받아 설치하셨겠죠? 다음으로 이 에러가 발생하는 이유는 JMF의 lib 디렉토리를 브라우져가 제대로 확인하지 못해서 NativeCode를 찾지 못하기 때문에 발생합니다. 이러한 라이브러리들은 Wlib 밑에 있구요. 이 디렉토리를 여러분의 bat 파일에 기록해 주어야 합니다. 하지만, 문제점 하나... 만약 IE를 사용하고 있고, 분명히 제대로 설정도 다 해주었는데.. 왜 안되는가 하는 질문에.. 답변은 가장 최신 버젼의 IE용 java VM을 다시 다운로드 받아야 한다는 것입니다. http://www.microsoft.com/java 이곳에서 다시 java VM을 다운로드 받아 설치하세요..

JMF 환경의 설정 방법

JMF는 많은 Media Handler , plug-ins , 그리고 캡쳐 디바이스들로 구성되어 있습니다. 이러한 구성 요소들은 사용자에 의해 설정가능하며 확장이 가능합니다. 시스템에 맞게 JMF의 환경구성을 재설정 해주기 위해서 JMF 설치에 함께 설치되는 JMFRegistry 프로그램을 이용하시면 됩니다.

JMF 구동

자. 이제까지 JMF를 구동하기 위해 필요한 패키지의 선택과 하드웨어 및 소프트웨어 사양에 대해서도 알아보았고, JMF를 설치하고 세팅하는 방법까지 모두 완료했습니다. 이제는 실제로 과연 JMF가 어떤 모습인지, 어떤 기능을 보여줄 것인지 직접 확인하는 순서입니다. 이제 실제로 구동을 해볼까요.

Java 프로그램이 애플릿과 애플리케이션으로 구분되는 것과 동일하게 JMF 프로그램 역시 자바의 애플릿과 애플리케이션 모든 방법으로 프로그래밍이 가능합니다. 또한 여러분이 작성한 프로그램은 JDK Applet Viewer, HotJava, Netscape, IE 등의 브라우져와 JMStduio 애플리케이션 프로그램을 통해서 그 기능과 동작을 확인할 수 있습니다.

또 한가지 주의하실 점이 있습니다. 실제로 많은 분들이 인터넷 브라우져를 통해서 JMF 프로그램을 구동할 때 접하는 문제점들인데 아래와 같은 방법을 이용하셔서 문제점을 해결하시기 바

랍니다.

JMF와 인터넷 브라우져간의 세팅방법

JMF와 Netscape 구동방법
JMF는 Netscape Communicator 4.06 이상의 버전에서만 구동이 됩니다. 먼저 버전확인을 하셔야 합니다. 또한 주의 하실점은 쉬운 설치를 위해서 JMF 설치전에 컴퓨터 시스템에 Netscape 가 먼저 설치되어 있어야 합니다. 이러한 경우 JMF는 인스톨 과정에서 Netscape의 존재를 검사하고, JMF가 Netscape에서 구동되도록 세팅을 해줍니다. 만약 JMF 설치시에 Netscape를 위한 설정이 알맞게 되지 않는 경우에는 아래와 같이 2가지 방법으로 여러분이 직접 설정을 해주어야 합니다.

방법 1:
먼저 JMF 2.1을 Uninstall 시키고 Netscape Communicator 와 JMF 2.1을 설치하셔야 합니다. 이때 주의하실점은 Netscape Communicator를 먼저 설치하시고, JMF 2.1을 후에 설치하셔야 합니다.

방법 2:
Netscape Communicator를 설치한후에 JMF Shared Libraries와 관련 클래스 파일을 Netscape 디렉토리에 복사하는 방법입니다.

%WINDIR%₩Java₩Classes directory 로부터 :
jmf.properties, soundbank.gm, jmf.jar and sound.jar 파일을
C:₩Program Files₩Netscape₩Communicator₩Program₩Java₩Classes 디렉토리에 복사합니다.

the %WINDIR%₩System directory 로부터:
jm*.dll and jsound.dll 파일을 C:₩ProgramFiles₩Netscape₩Communicator₩Program₩Java₩Bin 로 복사합니다.

한가지 주의하세요. 만약에 Netscape가 특정한 타입의 파일 형식을 인식하지 못한다면, 먼저 MINE 타입 설정을 확인해야 합니다. 기본적으로 Netscape는 MIDI (audio/midi), RMF (audio/rmf), and GSM (audio/x-gsm) files에 대한 세팅을 해주지 않습니다. 실제로 여러분이 직접 이 부분을 *Edit->Preferences* menu 를 통하여 추가해주셔야 합니다.

Running JMF with the JDK 1.2 Applet Viewer

실제로 JMF 를 이용하면서 가장 많이 듣는 질문입니다. JDK Applet Viewer 로는 자신이 작성한 프로그램을 구동하면 자바의 보안에러만 발생하는데.. 어떻게 해결해야 하나?? 이러한 문제점이 자주 발생합니다. 실제로 JDK 1.2 의 퍼미션을 통해서 애플릿 뷰어는 JMF 구동시에 Runtime Error 를 발생시킵니다. 이러한 문제는 JRE 에 퍼미션을 부여하는 policy 파일을 통해서 해결할 수 있습니다. 실제로 jmf 를 설치할 때 함께 설치되는 샘플 policy 파일을 이용하여 다음과 같이 해결할 수 있습니다.

```
appletviewer -J-Djava.security.policy=file:<policy file> <URL>
```

JMF 맛보기

혹시 설치 때문에 힘드시지는 않으셨나요? JMF는 자바의 확장 패키지이기 때문에 실제로 설치시에 주의할점이 많습니다. 아마도 위에서 언급한 설명을 차례로 따라오신 분이라면 큰 문제없이 설치하셨으리라 생각됩니다. 자. 그럼 이제 실제 JMF의 구동방법에 대해 알아보도록 하죠..

먼저 이론적인 JMF의 구조를 살펴보기 전에 간단한 JMF 프로그램으로 JMF의 구조적인 프로그래밍 방법에 대해 이야기 하겠습니다. 아래 예제는 JMF를 이용하는 애플릿 프로그램으로서 실제 구동에 필요한 전체 프로그램중에서 필요한 부분만 설명을 위해 작성한것입니다. 아래의 프로그램은 애플릿의 구동을 위해서 HTML 파일을 통해 원하는 파일을 입력받고 그 파일을 재생하는 프로그램입니다. 프로그램에 대한 자세한 설명은 뒤에서 하겠습니다. 먼저 JMF 프로그램의 골격과 함께 어떻게 자바 프로그램과 접목되는지 살펴보시고, 웹브라우져를 통해 어떻게 결과가 나타나는지에 대해 살펴보도록 하세요.

```
// PlayerApplet.html 파일의 내용
<html>
<applet code=PlayerApplet.class width=176 height=144>
<param name = FILE value = "LEAVE1.MOV" >
</param>
</applet>
</html>
```

```
// PlayerApplet.java 파일의 내용

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.media.*;
import java.net.*;

public class PlayerApplet extends Applet implements ControllerListener{
    Player player= null;

    public void init(){
        setLayout(new BorderLayout());
        String mediaFile = getParameter("FILE");
        try{
            URL url = new URL(getDocumentBase(), mediaFile);
            player = Manager.createPlayer(url);
            player.addControllerListener(this);
        }catch(Exception e){
        }
    }
    public void start(){
        player.start();
    }
    public void stop(){
        player.stop();
        player.deallocate();
    }
    public void destroy(){
        player.close();
    }
    public synchronized void controllerUpdate(ControllerEvent e){
        if (e instanceof RealizeCompleteEvent){
            Component component;
            if((component = player.getVisualComponent()) != null)
               add("Center", component);
            if((component = player.getControlPanelComponent()) != null)
```

```
                add("South", component);
                validate();
            }
        }
}
```

[그림 2] AppletPlayer 예제 프로그램의 구동 결과

JMF의 깊은 곳

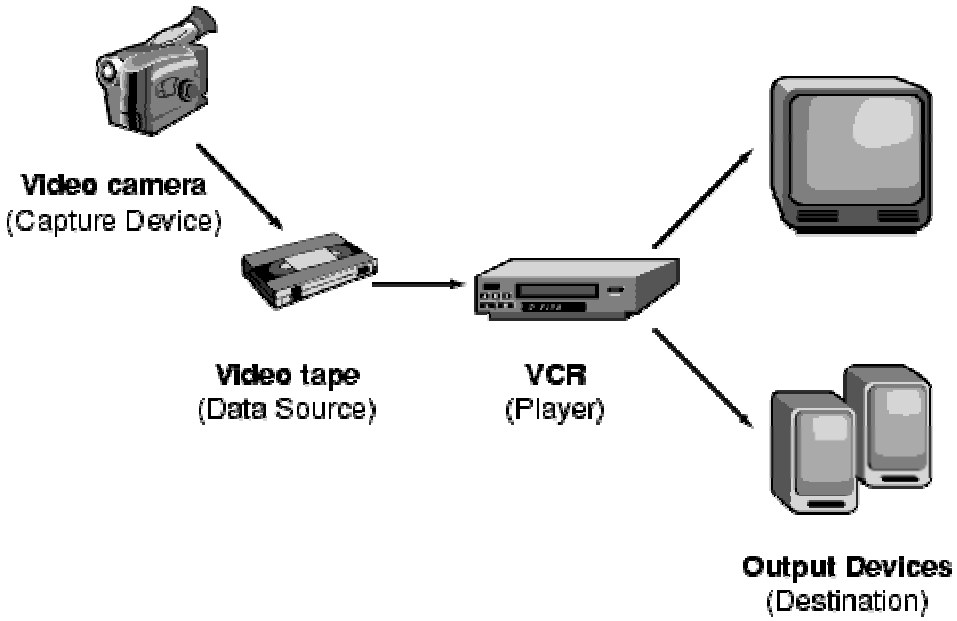
위에서는 애플릿으로 JMF를 이용한 간략한 동영상 파일 재생기를 만들어 보았습니다. 여러분이 직접보셨듯이 간단한 몇줄의 코드만을 이용해서도 [그림2] 에서와 같은 기능의 프로그램을 작성하실 수 있습니다. 자. 그럼 이러한 JMF의 기능을 이용하기 위해 JMF의 내부 구조를 살펴보는 여행을 떠나기로 하죠..

JMF의 깊은곳

이번에는 JMF에서 정의된 각종 클래스들의 개략적인 구성과 프로그래밍에 필요한 개념들, 그리고 관련 클래스와 이벤트처리 및 플레이어와 프로세서에 대한 설명을 하고자 합니다. 실제 프로그래밍은 없고 다소 소개적인 측면이 많지만, JMF라는 큰 모험을 시작하기 위한 준비단계라고 생각하시고 잘 읽고 이해해주시기 바랍니다.

1. High Level 구조

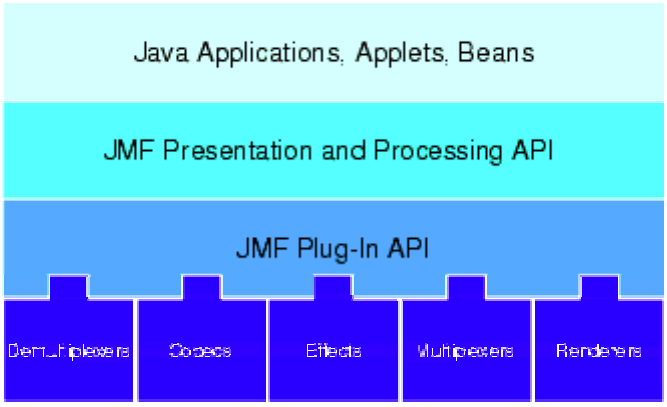
먼저 아래 [그림 3]을 한번 볼까요? 비디오 카메라로 영상을 얻어오고, 얻어온 영상을 비디오 테이프에 저장을 한후에 비디오 테이프를 비디오 플레이어에 넣고 나중에 TV를 통해서 본다는 아주 간단한 구조 입니다. 왜 이런 그림으로 소개를 했을까요.



[그림 3] JMF 클래스의 개념적인 구성도

바로 JMF의 내부구조가 바로 이러한 모습으로 정형화되어 있습니다. 위의 그림에서 비디오 테이프에 해당하는 부분이 바로 데이터 소스를 의미하고, 그림의 비디오 플레이어가 바로 JMF에서 미디어 파일을 연주하는 재생기능을 수행하는 역할을 합니다.

그럼 실제적인 High Level 구조를 지원하는 JMF의 내부구조를 살펴보도록 하죠. [그림 4]를 보아주세요.



[그림 4] JMF High Level 구조 설명

[그림 3] 및 [그림 4]에서처럼 데이터소스와 플레이어는 JMF의 High level API를 구성하며, 데이터의 캡쳐, 프리젠테이션, 그리고 time-based 미디어의 데이터 처리 역할을 담당합니다. 이러한 고수준의 API 제공뿐만 아니라 JMF는 low level의 API를 제공함으로서 Custom processing component와 extensions를 제공하는 기능을 수행하도록 설계되어 있습니다.

JMF의 기본 패키지 및 클래스 살펴보기

실제로 JMF 패키지에서는 아래와 같은 패키지들을 포함하고 있습니다.

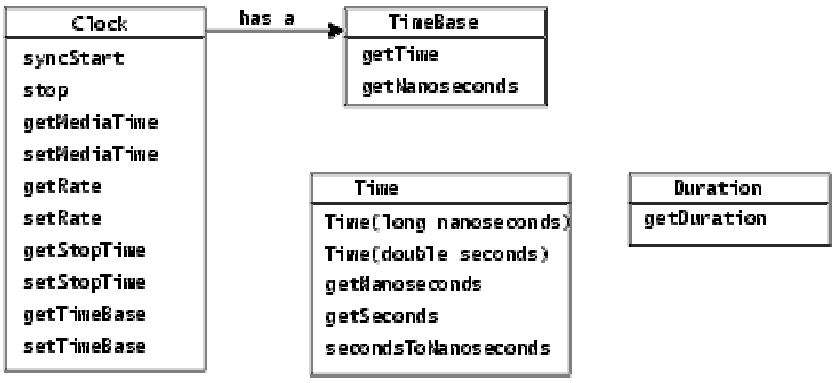
JMF 의 기본 Packages

javax.media
javax.media.bean.playerbean
javax.media.cdm
javax.media.control
javax.media.datasink
javax.media.format
javax.media.pim
javax.media.pm
javax.media.protocol
javax.media.renderer
javax.media.rtp
javax.media.rtp.event
javax.media.rtp.rtcp
javax.media.util

이러한 각 패키지 와 각 패키지들의 내부를 구성하는 클래스들을 전부 살펴볼 수는 없지만,

먼저 , JMF 프로그래밍을 위하여 기본적으로 알아야할 핵심 클래스들에 알아보도록 하죠.

JMF의 Clock 시스템



[그림 5] JMF의 Clock 구성 클래스관계

[그림 5]에서와 같이 JMF에서 멀티미디어 데이터를 다루기 위해서는 Clock, TimeBase 클래스를 이용합니다. 개념적으로는 TimeBase Time과 Media Time을 이용합니다. TimeBase Time은 TimeBase가 제공하는 시간으로서 우리가 일시 정지를 시키거나 멈출수 없는 실제 시간의 흐름을 관리하는 부분입니다. 이와는 다르게 Media Time은 미디어 데이터의 시작과 끝을 관리하는 시간입니다. 또한가지 Duration이라는 것은 Media Time에서의 경과 시간을 의미합니다. 실제로 멀티미디어 데이터가 플레이 될 때 Media Time은 아래와 같은 식에 의해 계산될 수 있습니다.

MediaTime = MediaStartTime + Rate(TimeBaseTime - TimeBaseStartTime)

쉬운 예제를 하나 들어볼까요? 우리가 사용하는 시계를 예로들면, 시계의 초침은 멈추거나 정지하지 않습니다. 계속 연속적인 흐름이지요. 이것이 바로 TimeBase로 대응되고, 스톱워치와 같이 시작과 끝이 존재하는 특정 시간이 MediaTime으로 대응이 됩니다. TimeBase의 인터페이스를 살펴보아도 쉽게 알수있는데 TimeBase에서는 getNanoSecond(), getTime()등 현재의 시간을 받는 함수만 제공할뿐, 시간을 멈추거나 정지시키는 함수는 제공하지 않습니다.

JMF의 Media Streams

Media Stream이라는 것이 무엇일까요. 미디어 스트림은 로컬파일, 네트워크를 통해 전송되어진 자료, 그리고 비디오나 오디오등의 캡쳐장치를 통해 획득된 미디어 데이터입니다. 예를들어 MPEG 파일이 비디오와 오디오데이터가 함께 존재하듯이 이러한 미디어스트림은 Track이라 불리우는 다채널데이타로 구성되어 있습니다.

미디어 스트림은 미디어 스트림의 위치정보와 이용되는 프로토콜정보에 의해 정의됩니다. 미디어 스트림에 대한 또하나의 분류 방법은 바로 미디어스트림의 전송 방식에 따른 분류입니다.

Pull Data transfer : 데이터 전송의 시작과 조정을 Client측에서 수행합니다.예를들면 HTTP, FILE 프로토콜등이 이 같은 예입니다.

Push Data transfer : Server측에서 데이터전송의 시작과 조정을 수행합니다. 예를들면 RTP(Realtime Transport Protocol)이나 실리콘그래픽사의 SGI MediaBase 프로토콜등이 있습니다.

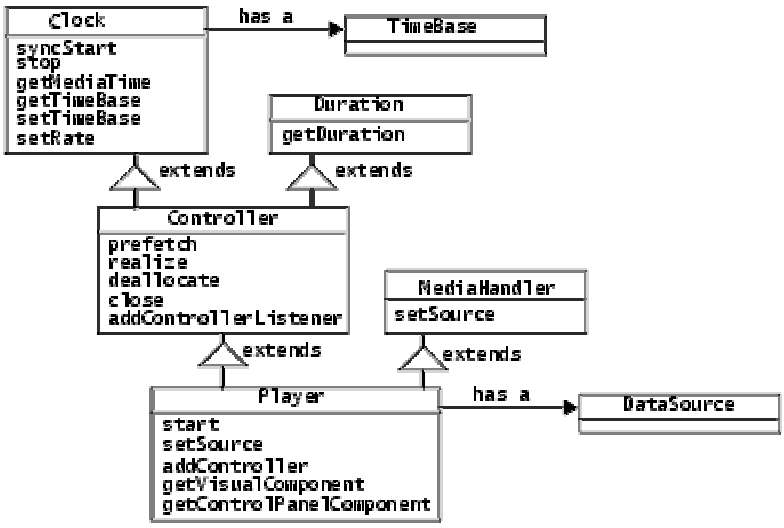
[그림 3]의 내용 기억나세요? [그림 3]에서보면 디지털비디오 레코더는 바로 Capture Device입니다. 그리고 그 자료는 비디오테이프에 저장되지요. 이러한 비디오 테이프의 개념이 바로 JMF이용하는 DataSource입니다. DataSource는 멀티미디어 데이터에 대한 위치정보, 프로토콜정보, 전송에 관한 정보를 나타내는 클래스입니다. JMF에서 이용하는 DataSource 역시 위와같이 PullDataSource와 PushDataSource를 이용합니다. 또한 버퍼의 이용유무에 따라서 Pull방식은 PullDataSource, PullBufferDataSource로 구분되고, Push방식은 PushDataSource, PushBufferDataSource로 구성됩니다. 이와 같은 기능이외에도 JMF에서는 Cloneable DataSource와 Merging DataSource를 제공합니다. Cloneable DataSource는 해당 데이터 소스를 복제하여 마치 여러 개의 데이터소스를 독립적으로 이용할수 있도록 하는 기능을 제공합니다. 예를들면 하나의 비디오테이프를 여러 개 복사하여 서로다른 비디오 플레이어에서 볼 수 있는 기능이라고 생각할 수 있습니다. Merging DataSource는 여러 개의 DataSource를 서로합쳐서 하나의 통합된 DataSource를 생성하는 기능입니다.

Player, Processor
자. 이번에는 [그림 3]에서 보였던 비디오테이프 플레이어에 대한 설명입니다. 비디오 테이프가 있더라도 실제로 이를 재생하기위한 플레이어가 필요합니다. JMF에서도 역시 이와 같은 개념으로 Player와 Processor를 제공합니다.

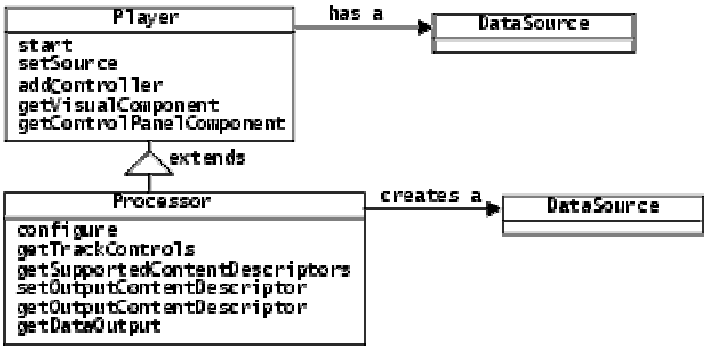
[그림 6]과 [그림 7]은 각각 Player와 Processor의 구조를 나타냅니다. 그렇다면 왜 이렇게 Player와 Processor로 분류를 한것일까요? [그림 3]에서처럼 비디오플레이어는 단지하나인데 왜 JMF에서는 Player, Processor를 구분한것일까요?

Player는 실제 미디어의 재생기능을 위주로 만들어진 클래스입니다. 예전에는 단순히 멀티미디어 데이터의 재생기능이 주된 관심사였기에 이렇게 Player로 재생을 하였습니다. 그러나, 재생되는 멀티미디어 데이터에 대하여 특정한 처리를 하고자할 때 예를들어, 데이터의

포맷변환이나 네트워크 전송, 파일로 저장등은 단순한 미디어 재생기능과는 다른 개념으로의 접근이 필요하게 되었습니다. 그렇게해서 확장된 것이 바로 Processor 입니다.

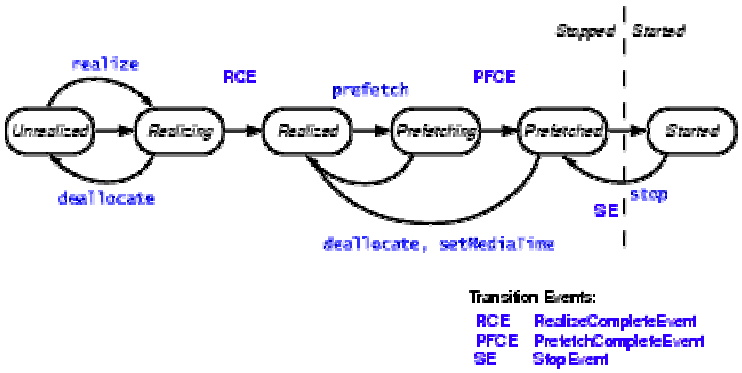


[그림 6] Player 구조

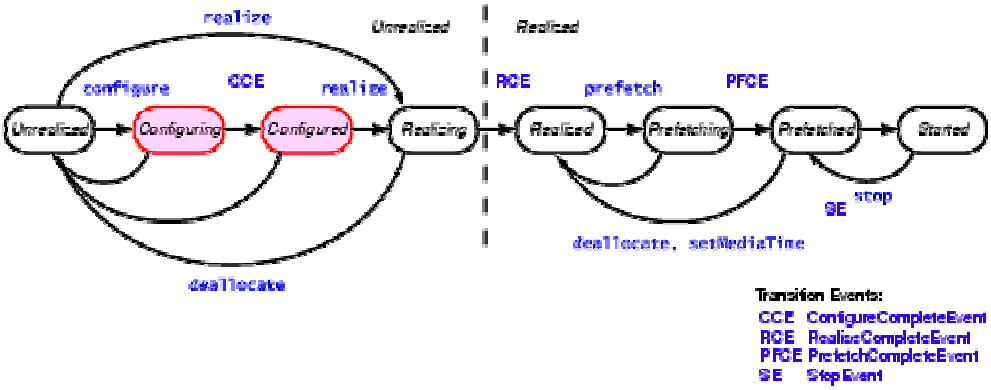


[그림 7] Processor의 구조

이글을 읽으면서 처음에 소개해드렸던 간단한 애플릿 예제를 살펴보셨나요? 일반적인 애플릿의 기본구조를 가지고 있지만, 특별한 이벤트 처리 부분이 있습니다. JMF에서는 Player와 Processor에 대한 프로그래밍을 하기위해서 현재 Player와 Processor가 어떤 상태에 있는지에 대한 상태정보를 이벤트화해서 프로그래머에게 이벤트 정보를 전달해줍니다. JMF 프로그램을 구동할 때 내부적으로 프로그램을 초기화하고, 관련 데이터소스를 얻어와 정보를 파악하고, 각종 컴포넌트 정보를 제공하고, 최종적으로 화면에 보여줄 준비를 하고, 마지막으로 화면에 보이는 작업을 합니다. 이러한 각 단계에 대해 JMF에서는 각 상태별로 이벤트가 발생되도록 정의하였습니다.



[그림 8] Player Status 구조도



[그림 9] Processor Status 구조도

[그림 8] 및 [그림 9]를 살펴보면 실제로 우리가 화면에서 멀티미디어 데이터를 재생하여 보기까지에는 상당히 많은 내부적인 단계를 거쳐야 하는 것을 알수있습니다. 그럼 각 단계별로 JMF는 내부적인 처리를 어떻게 하는지 Player를 기준으로 알아보도록 하죠.

Unrealized

- JMF 프로그램을 처음구동시키고 바로 이상태로 들어옵니다. 즉, Player 가 막 생성된 직후의 단계로서 이때 Player는 우리가 재생하고자하는 어떠한 멀티미디어 데이터에 대한 정보를 가지고 있지 않습니다. 단순히 Player라는 객체만 생성된 단계입니다.

Realizing

- Unrealized 상태에서 realize() 메쏘드가 호출되었을 때 이 상태로 이동합니다. 그러나 주의할점은 JMF에서 모든 상태로의 이동은 바로 호출되지 않고, 일정한 지연시간을 갖는다는 점입니다. 그러므로 우리가 프로그래밍을 할때에도 realize() 메쏘드를 호출한 후에 Realizing 상태로 이동했는지 반드시 확인하고 그에따른 작업을 수행해야 합니다.

Realized

- Realizing 상태로부터 RealizedCompleteEvent가 발생했을경우 이 상태로 진입합니다. 이 상태에서 Player는 재생하고자하는 멀티미디어 데이터에 대한 정보를 완전히 파악하고 있는 단계입니다. 이 단계에서 화면에 보이고자하는 각종 컴포넌트들의 정보를 획득할수 있습니다.

Prefetching

- Realized 단계에서 prefetch() 함수가 호출되면 이상태로 진입하게 됩니다. 이 상태에서는 현재재생중인 미디어 데이터의 시간변경 정보와 더불어 재생비율을 변경할때에도 이용됩니다.

Prefetched

- PrefetchingCompleteEvent가 발생되면 이 단계로 진입합니다. 실제로 이 단계에서는 모든 사전준비를 마치고, 완전하게 재생을 위하여 기다리는 단계가 됩니다.

Started

- 실제로 멀티미디어 데이터를 재생하는 단계입니다.

이젠 간략하게 Processor의 상태변화를 살펴보겠습니다. Processor에서는 Configuring 과 Configured 단계가 추가되었습니다. 이부분에 대한 설명은 아래와 같습니다.

Configuring

- Processor가 Unrealized 상태에서 configure() 메쏘드를 호출하면 이 상태에 진입합니다. 이상태에 있는 동안에 Processor는 DataSource에 연결을 시도하고, 입력되는 데이터 스트림에 대한 Demultiplexer를찾고 , 입력데이터의 포맷정보를 찾게 됩니다.

Configured

- ConfigureCompleteEvent가 발생되면 이 상태로 진입하게 됩니다. 이상태에서는 Processor가 완전히 DataSource에 연결이되었고, 내부적으로 이용할 데이터의 포맷이 결정되어진 상태입니다.

JMF 프로그래밍 만들기

이번 첫 강좌에서 정말 많은 이론적인 설명이 있엇습니다. JMF에 대한 소개와 설치, JMF의 주요기능과 클래스설명, 내부동작등도 살펴보았습니다. 이젠 직접 JMF를 이용한 프로그래밍

을 하는 부분입니다. 부지러한 독자들은 이미 강좌 앞부분에 나온 간단한 애플릿으로 구성된 JMF 프로그램을 실행하고 그 결과를 보셨을텐데요. 자. 한단계 한단계씩 차근히 설명하면서 프로그래밍을 해보도록 하죠.
이번에 만들 JMF 예제는 애플릿을 통하여 동영상 파일을 재생하는 프로그램입니다. 실제로 이 프로그램을 통하여 여러분은 거의 대부분의 동영상 파일 포맷을 재생할 수 있습니다. 자. 시작해볼까요.. 예제 프로그램은 자바 애플릿으로 JMF를 이용한 프로그램입니다. 당연히 HTML 파일과 Java 파일이 있어야 하겠지요. 먼저 예제 파일들과 컴파일 및 실행방법을 알아봅니다.

[예제 프로그램 이름]
SimplePlayerApplet.html
SimplePlayerApplet.java

[컴파일 및 실행방법]
javac SimplePlayerApplet.java
인터넷 브라우져에서 SimplePlayerApplet.html 파일을 읽기

[그림 10] SimplePlayerApplet의 실행 화면

우리가 처음에 간략하게 살펴보았던 예제와 모습은 동일하죠? 하지만, 이번 예제에서는 애

플릿에서 JMF를 구동하는 방법과 더불어서 JMF의 Player가 여러단계의 상태변화를 통해 어떤 메시지를 전달하고 우리는 어떻게 프로그래밍을 할것인가 즉, 이벤트 처리부분에 대한 좀더 세부적인 사항을 설명하려고 합니다. 자. 그럼 소스 코드를 보면서 설명을 하겠습니다. 먼저 HTML 소스파일부터 살펴보도록 하죠.

```
// SimplePlayerApplet.html
<html>
<title> JMF Programming Test Program
</title>
<applet code=SimplePlayerApplet width=320 height=300>
<param name=file value="bluescreen2.mov">
</applet>
</html>
```

위의 소스코드에서 보시듯이 일반 자바 애플릿 프로그램과 다른점이 없습니다. 한가지 주의할점은 param 형식으로 플레이할 파일의 이름을 적어서 value값에 주면 자바 프로그램에서 이를 읽어들이는 부분이 있습니다.

```
<param name=file value="bluescreen2.mov">
```

이 부분에서 여러분이 원하시는 파일 이름으로 value의 값을 바꾸어주시면 됩니다. 물론 플레이를 하고자 하는 파일과 HTML파일 및 SimplePlayerApplet 클래스 파일은 동일한 디렉토리에 있어야 합니다.

이번에는 실제 SimplePlayerApplet.java 파일을 살펴보도록 하죠.

HTML의 param을 이용한 파라미터 입력
먼저 프로그램에서는 HTML의 param 형태로 읽은 미디어데이터 파일이름을 얻어와야 합니다. 이 부분은 아래와 같습니다.

```
if ((mediaFile = getParameter("FILE")) == null)
    Fatal("Invalid media file parameter");
    try {
        url = new URL(getDocumentBase(), mediaFile);
        mediaFile = url.toExternalForm();
```

```
    } catch (MalformedURLException mue) {
}

void Fatal (String s) {
    System.err.println("FATAL ERROR: " + s);
    throw new Error(s);
 }
```

소스에서 보듯 HTML의 file 파라미터의 value 값을 String으로 읽어들입니다. 그 값이 없을 경우 해당에러 처리를 하고 프로그램을 종료합니다. 이렇게 얻은 파일이름과 HTML 도큐먼트의 정보로부터 URL을 만들었습니다.

다음으로 해야할 작업은 MediaLocator의 정보를 생성하는 부분입니다. MediaLocator에는 미디어데이터에 대한 위치정보가 들어가게 됩니다. 아래에서 보시는 것 처럼 URL정보로부터 MediaLocator를 생성합니다.

```
if ((mrl = new MediaLocator(mediaFile)) == null)
    Fatal("Can't build URL for " + mediaFile);
```

Player를 만드는 부분

위와 같은 과정을 통해서 미디어데이터에 대한 위치정보를 얻어왔습니다. 이제는 Player를 만들어야합니다. Player를 만드는 방법은 Manager 클래스를 이용하여 아래와같이 3가지의 방법중 하나를 이용하게 됩니다.

```
public static Player createPlayer(MediaLocator sourceLocator)
                            throws java.io.IOException,
                                   NoPlayerException

public static Player createPlayer(java.net.URL sourceURL)
                            throws java.io.IOException,
                                   NoPlayerException

public static Player createPlayer(DataSource source)
                            throws java.io.IOException,
                                   NoPlayerException
```

위의 3가지 방법을 간략히 살펴보면, 데이터의 URL 정보로 player를 생성하는 방법, DataSource로 player를 생성하는 방법, 그리고 예제 프로그램에서 이용한 MediaLocator를 통해 player를 생성하는 방법등이 있습니다. 자. 그럼 실제 프로그램에서 적용한 방법을 보도록 하죠.

```
try {
        player = Manager.createPlayer(mrl);
 } catch (NoPlayerException e) {
        System.out.println(e);
        Fatal("Could not create player for " + mrl);
 }
```

Manager클래스의 createPlayer를 이용하여 Player를 생성하였으며, 3가지의 Player 생성방법중에서 MediaLocator를 이용하였습니다.

Player의 이벤트처리 구조

이제는 생성된 Player에 대하여 각종 해당되는 이벤트를 처리하는 부분입니다. 일반적인 자바 프로그램과 비슷한 방법으로 JMF에서는 아래와 같은 방법을 이용합니다.

```
implements ControllerListener
player.addControllerListener(this);
```

첫째, 우리가 만든 자바애플릿은 ControllerListener를 구현상속해야 합니다. 즉, Player의 모든 이벤트처리는 바로 ControllerListener를 상속받아 해당 이벤트 처리를 해주어야 합니다. 둘째로 생성된 Player에게 이벤트 처리를 등록시켜야 합니다. 마지막으로 이렇게 ControllerListener 이벤트를 처리할 때 반드시 구현해야 하는 메쏘드가 있습니다.

```
public synchronized void controllerUpdate(ControllerEvent event) {
}
```

위와같이 메쏘드의 이름은 controllerUpdate이고, 이 메쏘드의 인자인 ControllerEvent의 종류에 따라서 Player의 상태를 알수 있고, 상태에 따른 해당작업을 수행할 수 있습니다. 그럼 이와 같은 controllerUpdate 메쏘드내부에서의 이벤트 처리과정을 살펴보도록 하죠.

```
if (event instanceof RealizeCompleteEvent) {
        //
} else if (event instanceof CachingControlEvent) {
        //
} else if (event instanceof EndOfMediaEvent) {
        //
} else if (event instanceof ControllerErrorEvent) {
        //
}
```

controllerUpdate 메쏘드의 파라미터인 ControllerEvent는 위와같이 instanceof를 이용하여 이벤트의 종류를 판별할수 있습니다. 이외에도 많은 이벤트들이 있으니 여러분이 직접 API 문서를 참조하시기 바랍니다.

애플릿의 기본동작과 Player

애플릿 프로그램의 기본적인 메쏘드인 init(), start(), stop(), destroy()등의 메쏘드 내부에서 Player를 제어할 수 있습니다. 애플릿이 시작하는 start() 메쏘드 수행시에 Player를 시작시키고, 애플릿이 중단되는 stop()메쏘드에서 Player를 중단시키며, 애플릿이 종료되는 destroy() 메쏘드내에서 Player를 종료시킬수 있습니다. 아래와 같은 방법으로 구현을 할 수 있습니다.

```
public void start() {
            if (player != null)
            player.start();
}

public void stop() {
        if (player != null) {
        player.stop();
        player.deallocate();
    }
}

public void destroy() {
        player.close();
}
```

JMF의 Component 얻어오기

JMF의 컴포넌트들은 Player가 완전히 Realized 상태, 즉 RealizeCompleteEvent 가 발생한후에 이용할 수 있습니다. 이러한 컴포넌트는 Visual Component와 ControlPanel Component가 있으며, 이러한 컴포넌트들을 획득한후에 적절한 Layout Manager를 이용하여 원하는 위치에 표현할 수 있습니다. Component를 얻는 메쏘드는 2가지 종류가 있습니다.

```
player.getControlPanelComponent()
player.getVisualComponent()
```

위와 같은 메쏘드를 이용합니다. 실제 이부분에 대한 구현 소스는 아래와 같습니다.

```
if (event instanceof RealizeCompleteEvent) {
        int width = 320;
        int height = 0;
        if (controlComponent == null)
        if (( controlComponent = player.getControlPanelComponent()) != null) {
            controlPanelHeight = controlComponent.getPreferredSize().height;
            panel.add(controlComponent);
            height += controlPanelHeight;
        }


        if (visualComponent == null)
                if (( visualComponent = player.getVisualComponent())!= null) {
                        panel.add(visualComponent);
                        Dimension videoSize = visualComponent.getPreferredSize();
                        videoWidth = videoSize.width;
                        videoHeight = videoSize.height;
                        width = videoWidth;
                        height += videoHeight;
                        visualComponent.setBounds(0, 0, videoWidth, videoHeight);
                }
        panel.setBounds(0, 0, width, height);
        if (controlComponent != null) {
                        controlComponent.setBounds(0,       videoHeight,       width,
```

```
controlPanelHeight);
                        controlComponent.invalidate();
        }
        }
}
```

대용량 파일의 다운로드 과정표기

실제로 웹에서 대용량의 멀티미디어 파일을 플레이하기 위해서 다운로드를 받는경우, 현재 어느정도까지 다운로드를 받았는지, 앞으로 어느정도의 시간이 더 필요한지에 대한 정보를 시각적으로 제공해준다면 유저가 프로그램을 이용하기에 한결 편해지겠지요. JMF에서도 이러한 방법을 제공해 줍니다. ControllerEvent의 종류중에서 CachingControlEvent 를 통해서 접근을 할 수 있으며, 획득한 이벤트의 getCachingControl() 메쏘드를 통하여 다운로드 컨트롤을 얻을 수 있습니다. 실제 구현된 소스는 아래와 같습니다.

```
if (event instanceof CachingControlEvent) {
    if (player.getState() > Controller.Realizing)
                        return;
           // 다운로드 하는경우 ProgressBar를생성하고
           // 다운로드 종료시 ProgressBar 제거
           CachingControlEvent e = (CachingControlEvent) event;
           CachingControl cc = e.getCachingControl();

           if (progressBar == null) {
                if ((progressBar = cc.getControlComponent()) != null) {
                   panel.add(progressBar);
                   panel.setSize(progressBar.getPreferredSize());
                   validate();
                 }
          }
}
```

미디어의 반복 재생부분

미디어 파일의 재생을 모두 마치면, 기본적으로 JMF는 반복재생을 하지 않습니다. 그렇기 때문에 미디어 데이터의 재생종료를 알리는 이벤트 처리를 해주어야 합니다. 이렇듯 재생종료시 발생하는 이벤트는 EndOfMediaEvent 입니다. 이 이벤트가 발생했을 때 Player의 MediaTime을 0 값, 즉 최초의 시작시간으로 옮겨주고 다시 Player를 시작시키면 미디어데이

타의 재생이 종료되는 마지막시점에서 다시 처음부터 재생을 시작하게 됩니다. 아래는 구현된 소스입니다.

```
if (event instanceof EndOfMediaEvent) {
            player.setMediaTime(new Time(0));
            player.start();
}
```

Controller의 상태처리에 관한 이벤트
Controller의 특정 상태처리에 관한 이벤트처리도 가능합니다. 이때에는 ControllerCloseEvent 및 ControllerErrorEvent 이벤트를 처리해주면 됩니다. 관련된 소스는 아래와 같습니다.

```
if (event instanceof ControllerErrorEvent) {
        player = null;
        Fatal(((ControllerErrorEvent)event).getMessage());
} else if (event instanceof ControllerClosedEvent) {
            panel.removeAll();
  }
}
```

이상과 같이 전체적인 프로그램 소스에 대하여 기능별로 분석을 시도했습니다. 아래에 전체 예제의 소스를 게제합니다. 실제로 구동을 해보시고, JMF의 다양한 기능에 대해서도 살펴보도록 하세요.

```
// 애플릿 구동을 위한 패키지 임포트 부분
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.lang.String;
import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;
import java.util.Properties;
```

```
// JMF 구동을 위한 패키지 임포트 부분
import javax.media.*;

// 애플릿 기반의 JMF 프로그램
// Player의 상태변화 이벤트 처리를 위하여 ControllerListener 구현상속
public class SimplePlayerApplet extends Applet implements ControllerListener {
    // 미디어 데이터의 플레이를 위한 객체 : Player
    Player player = null;

    // 미디어 데이터의 컴포넌트 선언
    Component visualComponent = null;
    Component controlComponent = null;

    // 미디어데이터를 다운로드 받을때 표시되는 ProgressBar
    Component progressBar = null;


    // 화면 구성에 관계되는 변수들
    boolean firstTime = true;
    long CachingSize = 0L;
    Panel panel = null;
    int controlPanelHeight = 0;
    int videoWidth = 0;
    int videoHeight = 0;


    // 애플릿 초기화 부분
    public void init() {
                // 레이아웃 결정
                        setLayout(null);
                        setBackground(Color.white);
                        panel = new Panel();
                        panel.setLayout( null );
                        add(panel);
                        panel.setBounds(0, 0, 320, 240);
                        String mediaFile = null;
                        MediaLocator mrl = null;
```

```
                URL url = null;

                // HTML의 param의 value 값을 읽고
                if ((mediaFile = getParameter("FILE")) == null)
      Fatal("Invalid media file parameter");
                try {
                // URL정보를 생성
                url = new URL(getDocumentBase(), mediaFile);
                mediaFile = url.toExternalForm();
                } catch (MalformedURLException mue) {
                }

            try {
        // URL 정보를 통하여 MediaLocator 생성
        if ((mrl = new MediaLocator(mediaFile)) == null)
                        Fatal("Can't build URL for " + mediaFile);

        try {
                        // MediaLocator로부터 Player 생성
                        player = Manager.createPlayer(mrl);
        } catch (NoPlayerException e) {
                        System.out.println(e);
                        Fatal("Could not create player for " + mrl);
        }

        // Player의 이벤트 리스너 등록
        player.addControllerListener(this);

            } catch (MalformedURLException e) {
        Fatal("Invalid media file URL!");
            } catch (IOException e) {
        Fatal("IO exception creating player for " + mrl);
            }
}

// 애플릿 구동시에 player 시작되도록
```

```
public void start() {
  if (player != null)
         player.start();
}

// 애플릿 정지시에
public void stop() {
    if (player != null) {

        // Player를 정지시키고 할당되어진 리소스를 풀어줌
        player.stop();
        player.deallocate();
    }
}

public void destroy() {
                    // Player 닫음
                    player.close();
}

// Player의 이벤츠 처리 메쏘드 구현 부분
public synchronized void controllerUpdate(ControllerEvent event) {
                    if (player == null)
            return;

                    // Player가 Realized된 상태
                    // 이때 Component를 얻어올수 있다.
                    if (event instanceof RealizeCompleteEvent) {
            // 다운로드시에 이용된 ProgressBar 제거
            if (progressBar != null) {
                                panel.remove(progressBar);
                                progressBar = null;
            }
            int width = 320;
            int height = 0;
```

```
                // ControlPanelComponent 얻기
                if (controlComponent == null)
                                if (( controlComponent =
                      player.getControlPanelComponent()) != null) {

                        controlPanelHeight                                          =
ontrolComponent.getPreferredSize().height;
                        panel.add(controlComponent);
                        height += controlPanelHeight;
                                }

                                // VisualComponent 얻기
                if (visualComponent == null)
                                        if (( visualComponent =

                        player.getVisualComponent())!= null) {

                                panel.add(visualComponent);

                                Dimension               videoSize               =
visualComponent.getPreferredSize();

                                videoWidth = videoSize.width;

                                videoHeight = videoSize.height;

                                width = videoWidth;

                                height += videoHeight;

                                visualComponent.setBounds(0,      0,      videoWidth,
videoHeight);

                                        }



                        // 얻어진 컴포넌트들을 부착시킨다.
```

```
                    panel.setBounds(0, 0, width, height);
                    if (controlComponent != null) {
                                                controlComponent.setBounds(0,
videoHeight,
                                            width, controlPanelHeight);
                                            controlComponent.invalidate();
                    }
            } else if (event instanceof CachingControlEvent) {

                // 다운로드 진행시의 이벤트 처리
                if (player.getState() > Controller.Realizing)
                                        return;

                CachingControlEvent e = (CachingControlEvent) event;
                CachingControl cc = e.getCachingControl();

                // 다운로드 경과정도를 ProgressBar의 위치로 표현함
                if (progressBar == null) {
                if ((progressBar = cc.getControlComponent()) != null) {
                                panel.add(progressBar);
                                panel.setSize(progressBar.getPreferredSize());
```

```
                        validate();
                                }
            }
                    } else if (event instanceof EndOfMediaEvent) {
            // 미디어 재생을 끝까지 마친경우 재생위치를 처음으로 되돌림
            player.setMediaTime(new Time(0));
            player.start();
                    } else if (event instanceof ControllerErrorEvent) {
            // Error 발생시
            player = null;
            Fatal(((ControllerErrorEvent)event).getMessage());
    } else if (event instanceof ControllerClosedEvent) {
            // Controller 종료시에
            panel.removeAll();
                    }
}

 void Fatal (String s) {
                    // 예기치 못한 에러발생시 종료
                    System.err.println("FATAL ERROR: " + s);
                    throw new Error(s);
```

```
    }

}
```

마치면서

이번 강좌에서는 JMF에 대한 소개와 상세한 설치과정 및 세팅방법과 애플릿기반의 JMF 플레이어에 대한 프로그래밍과 이벤트 처리에 대하여 알아보았습니다. 다양한 멀티미디어 데이터 포맷을 지원하며, 그 기능이 점점 더 향상되고 있는 JMF에 대해 좀더 쉽게 접근할 수 있는 계기가 되었으면 합니다. 다음 강좌에서는 Swing과 JMF를 결합하여 Preview기능을 가지는 애플리케이션의 작성에 대한 강좌가 진행됩니다.

강좌 관련 소스파일

SimplePlayerApplet.html

SimplePlayerApplet.java

필자 연락처 : kingseft@samsung.co.kr