

# Использование Mobile Media API (JSR 135) – Часть 1

Автор: Мартин де Джод (Martin de Jode)

Перевод: Дмитрий Соколов (Dmitry Sokolov)

Published by the Symbian Developer Network

Статус: версия 1.2

Дата: 6 декабря 2003

|   |  |   |
|---|--|---|
| 1 | ВВЕДЕНИЕ .....                                       | 2 |
| 2 | MOBILE MEDIA API .....                               | 2 |
| 3 | АРХИТЕКТУРА MOBILE MEDIA API .....                   | 2 |
| 4 | ВОСПРОИЗВЕДЕНИЕ ВИДЕО – ПРОСТЕЙШИЙ ВИДЕО ПЛЕЕР ..... | 5 |
| 5 | ЗАКЛЮЧЕНИЕ .....                                     | 8 |
| 6 | РЕСУРСЫ .....  | 9 |

## 1 Введение

Данный документ – первый из двух, описывающих возможности Mobile Media API. В этом документе мы дадим общее представление о Mobile Media API и его архитектуре. Затем мы проиллюстрируем несколько представленных идей, более подробно рассмотрим поддержку воспроизведения видео, детально изучив простой видео плеер. Во втором документе речь пойдет о воспроизведении аудио, генерации тона и захвате изображения.

## 2 Mobile Media API

Mobile Media API (JSR 135) – дополнительное API для J2ME устройств на базе CLDC, поддерживающих воспроизведение и запись аудио и видео, а также захват изображения со встроенной камеры и генерацию тона. Будучи нацеленным на CLDC-устройства, MMAPI тем не менее может использоваться и в других средах (например, CDC), но CLDC берётся как наименьший общий знаменатель.

Цель JSR 135 – предоставить поддержку широкого круга возможностей для различных устройств, возможно не реализуя весь функционал, заявленный в спецификации. Поскольку MMAPI является опциональным пакетом, нельзя гарантировать поддержку редких типов медиа данных или протоколов, кроме тех, которые предопределены в API в связи с особенностями аппаратной реализации.

Не смотря на то, что Mobile Media API не гарантирует поддержку каких-либо специфических типов медиа данных, требуется, чтобы реализация поддерживала данный тип медиа в виде элементов управления. Следовательно, реализация поддержки видео должна предоставляться с помощью VideoControl, а реализация генерации тонов – с помощью ToneControl. Спецификация также описывает возможности, которые должны быть включены в реализацию (т.е. рекомендует). Например, реализацию поддержки аудио медиа данных должны предоставлять AudioControl и StopTimeControl; поддержка видео должна включать FramePositioningControl, StopTimeControl и VolumeControl (если аудио также доступно). Другие элементы управления, такие как RecordControl и RateControl, полностью опциональны. Полный список этих требований см. в [спецификации](#) MMAPI.

Также стоит упомянуть о том, что спецификация MIDP 2.0 (JSR 118) поддерживает лишь те типы аудио, которые являются подмножеством Mobile Media API, включая поддержку для

- Воспроизведение аудио

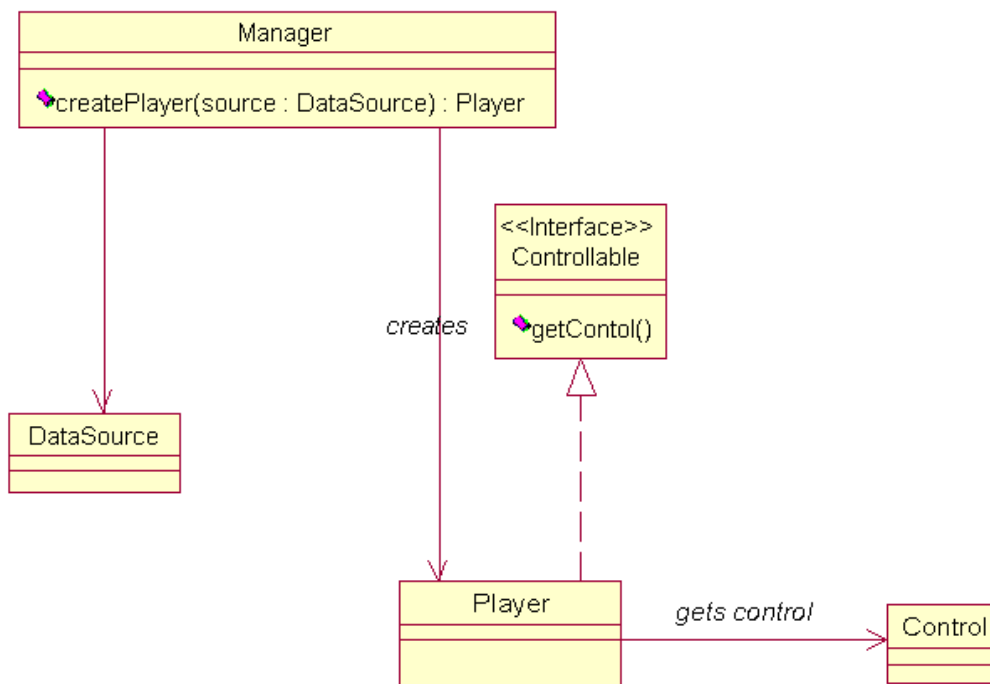
- Управление громкостью

- Генерация тона

Текущая версия Symbian OS, версия 7.0, включает MIDP 2.0 и из этого следует, что аудио является подмножеством Mobile Media API. Более того, камерофон Nokia 3650 на базе Symbian OS предоставляет расширенную поддержку Mobile Media API, реализованную Nokia, и строго следующую JSR 135.

## 3 Архитектура Mobile Media API

Простейшая архитектура Mobile Media API показана ниже




---

**Рисунок 1 Архитектура Mobile Media API**

---

Экземпляры класса `Player` создаются с помощью фабричного метода `createPlayer(...)`, который имеет следующую сигнатуру

```

createPlayer(java.lang.String locator)
createPlayer(java.io.InputStream stream, java.lang.String type)
createPlayer(DataSource source)
  
```

Первый метод `createPlayer(...)` получает в качестве параметра медиа локатор (*media locator*) типа `String`. Синтаксис локатора имеет несколько форм, примеры которых показаны ниже.

```

http://www.myserver.com/myvideo.mpeg
capture://audio
capture://video
capture://video?encoding=gray8&width=160&height=120
  
```

Синтаксис локатора захвата (*capture locator*) представлен в форме Бэкуса-Наура (БНФ, *Augmented Backus-Naur Format, BNF*) и более детально описывается в спецификации *Mobile Media API*.

Второй вариант метода `createPlayer(...)` принимает `InputStream` и тип MIME в качестве аргументов, и предназначен, например, для чтения потока байт из хранилища RMS (*RMS store*)

```

Player p = Manager.createPlayer(inputStream, "video/mpeg");
  
```

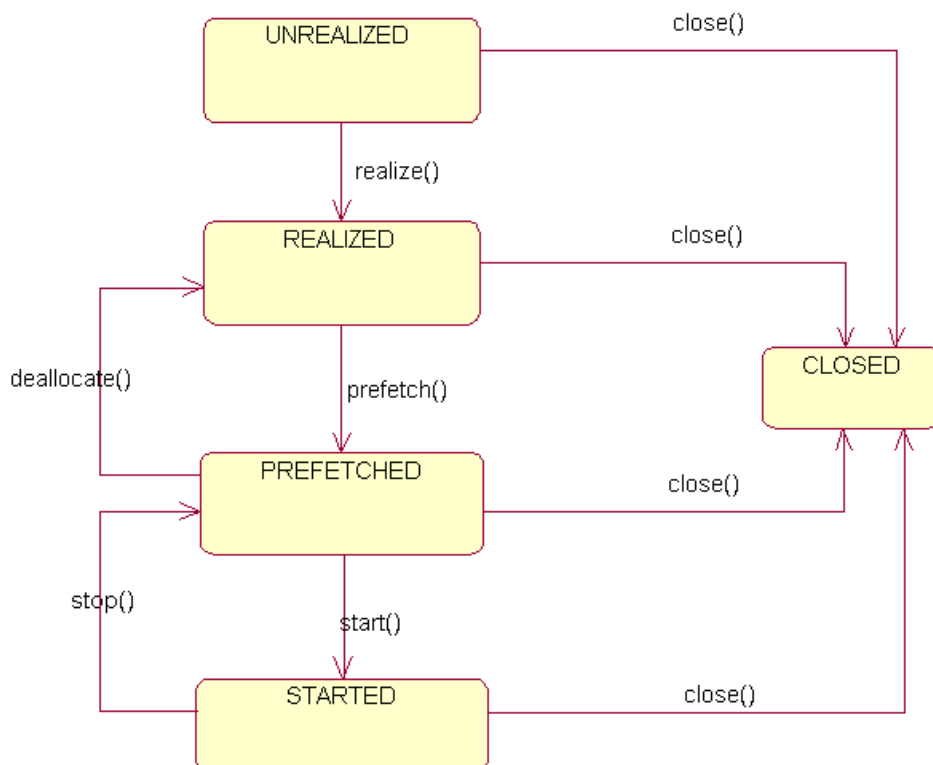
Последний метод принимает произвольный источник данных, производный от абстрактного класса `DataSource`, и предназначен для протоколов, определённых в приложении.

```

Player p = Manager.createPlayer(customDataSource);
  
```

Следует заметить, что все варианты `createPlayer(...)` вызывают исключение `MediaException` в случае, если невозможно создание плеера данного типа (например, если этот тип медиа не поддерживается). Они также вызывают исключение `IOException`, если имеет место проблема соединения или чтения из потока.

`Player` – высокоуровневая структура, которая управляет визуализацией протяжённых во времени медиа данных. Жизненный цикл `Player` показан на следующей диаграмме.



**Рисунок 2 Жизненный цикл плеера**

Только что созданный плеер находится в состоянии `UNREALIZED`. Метод `realize()` выполняет соединения со всеми необходимыми ресурсами (например, связь с сервером или файловой системой). Когда метод возвращает управление, плеер переходит в состояние `REALIZED`. Вызов метода `prefetch()` запрашивает ограниченные и специальные ресурсы, необходимые для воспроизведения медиа, такие как доступ к звуковому устройству телефона. Возможно, плееру придётся ждать освобождения этих ресурсов другим приложением, перед тем как перейти в состояние `PREFETCHED`. Только в этом состоянии плеер готов к запуску. Для выяснения состояния `Player`'а предоставлен метод `getState()`.

Прежде чем медиа данные могут быть воспроизведены, необходимо создать хотя бы один объект `control`. Объекты `control` используются для управления обработкой медиа и доступны из `controllable`, в данном случае из `Player`, используя метод `getControl(String controlType)`. Как было сказано в предыдущем разделе, медиа плеер данного типа может поддерживать произвольное количество `control`'ов. Метод `getControls()` служит для определения доступных управляющих структур. Заметим, что методы `getControl(...)` и `getControls()` плеера могут вызываться только тогда, когда тот находится в состоянии `REALIZED`, иначе будет вызвано исключение `IllegalStateException`.

Последняя тема этого короткого введения в Mobile Media API – интерфейс `PlayerListener`. Интерфейс `PlayerListener` предоставляет метод `playerUpdate(...)` для получения асинхронных событий от `Player`'а. Класс может реализовывать этот интерфейс и регистрировать `PlayerListener`, используя метод `addPlayerListener(...)` класса `Player`. `PlayerListener` реагирует на стандартные предопределённые события, включая `STARTED`, `STOPPED`, `END_OF_MEDIA`. Список стандартных событий может быть найден в [документации](#) к ММАРІ.

## 4 Воспроизведение видео – простейший видео плеер

Сейчас мы проиллюстрируем несколько концепций, представленных в предыдущем разделе, примерами кода из простейшего видео плеера `MIDlet`. `MIDlet` состоит из двух частей: `videoPlayer`, который наследуется от `MIDlet` и имеет атрибут `Form` для отображения `TextField` и `Gauge`, плюс некоторые элементы управления ('Play,' 'Exit'); класс `videoCanvas` для отображения видео. Скриншоты мидлета, запущенного на Nokia 3650, показаны ниже.

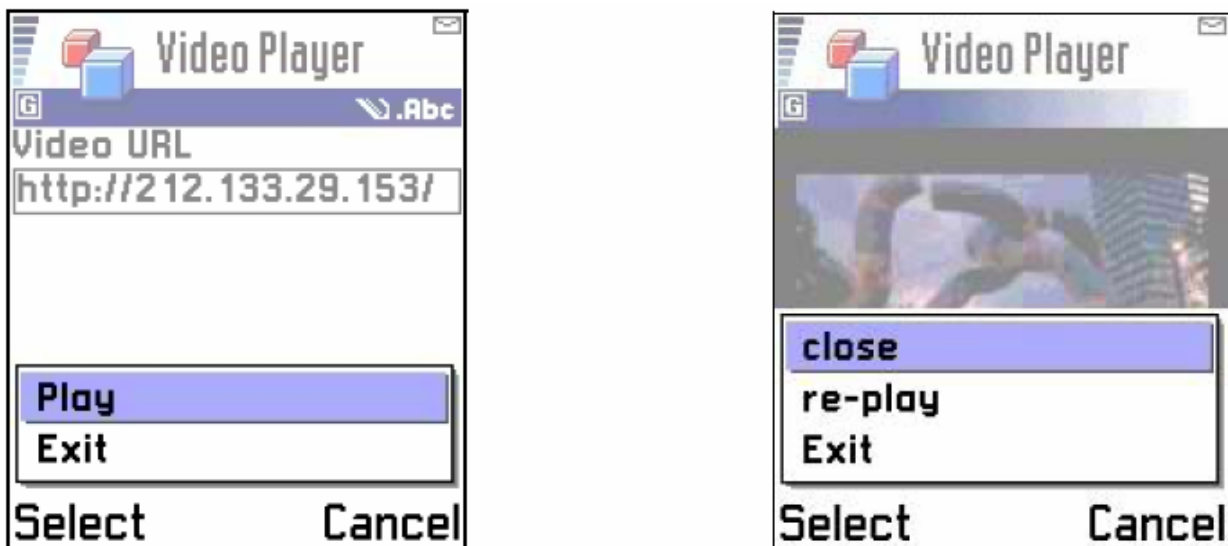


Рисунок 3 Скриншоты простейшего мидлета, воспроизводящего видео

Мы сконцентрируем своё внимание на классе `videoCanvas`, поскольку он содержит код, вызывающий Mobile Media API. Объявление класса `videoCanvas` приведено ниже.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import java.io.*;
```

```
public class videoCanvas extends Canvas implements CommandListener, PlayerListener,
Runnable
```

```
public videoCanvas(videoPlayer parent){}
```

```

public void initializevideo(String url){}

public void run(){}

public void playvideo(){}

public void paint(Graphics g){}

public void playerUpdate(Payer p, String event, Object eventData){}

public void commandAction(Command c, Displayable s){}

}

```

Класс `VideoCanvas` является производным от `Canvas`, а также реализует интерфейсы `PlayerListener`, `Runnable` и `CommandListener`.

Рассмотрим более подробно ключевые методы. Представленный ниже конструктор совершает стандартную инициализацию.

```

public VideoCanvas(VideoPlayer parent){
    super();
    this.parent = parent;
    display = Display.getDisplay(parent);
    close = new Command("close", Command.SCREEN, 1);
    addCommand(close);
    setCommandListener(this);
}

```

Метод `initializevideo(...)` принимает в качестве аргумента URL видео и затем запускает новый `Thread` чтобы выполнить необходимую инициализацию для воспроизведения видео данных.

```

public void initializevideo(String url){
    this.url = url;
    initializer = new Thread(this);
    initializer.start();
}

```

Метод `run()`, объявленный интерфейсом `Runnable`, содержит инициализацию `Player`'а. Есть много причин по которым рационально совершать инициализацию в отдельном потоке. Может быть желательно выполнять потенциально протяжённую во времени инициализацию в фоновом режиме, так как отображение видео требует старта с минимальной задержкой. Другой повод выполнять инициализацию в отдельном потоке – возможность создания индикатора прогресса, который давал бы пользователю представление о том, как идёт загрузка видео. Так как `VideoCanvas` реализует интерфейс `PlayerListener`, мы регистрируем экземпляр `Player` для приёма обратной связи. Методы `prefetch()` и `realize()` вызываются при обновлении плеера и индикатора загрузки.

```

public void run() {
    try{
        player = Manager.createPlayer(url);
        parent.updateGauge();
        player.addPlayerListener(this);
        player.realize();
        parent.prefetch();
        parent.updateGauge();
    }catch (IOException ioe){
        //обработка
    }
}

```

```

    }catch (mediaException me){
        //обработка
    }
    playVideo();
}

```

Пока плеер находится в состоянии PREFETCHED, мы можем визуализировать видео данные. В этом примере наш метод playVideo() вызывается немедленно.

```

public void playVideo() {
    try{
        videoControl = (videoControl)Player.getControl("videoControl");
        if(videoControl != null ){
            videoControl.initDisplayMode(videoControl.USE_DIRECT_VIDEO, this);
        }

        parent.updateGauge();

        int cHeight = this.getHeight();
        int cWidth = this.getWidth();
        videoControl.setCurrent(this);
        player.start();
    }catch (MediaException me) {
        //обработка
    }
}

```

Метод playVideo() управляет визуализацией видео на Canvas. Для того, чтобы сделать это, мы должны получить videoControl, вызвав getControl(...) на Player'e, находящемся в состоянии REALIZED.

Метод initDisplayMode(...) используется для инициализации видеорежима, в котором будет показано видео. Этот метод принимает целое число – одно из двух предопределённых значений USE\_GUI\_PRIMITIVE или USE\_DIRECT\_VIDEO – в качестве первого аргумента. В случае реализации MIDP (поддерживающей LCDUI) USE\_GUI\_PRIMITIVE вернёт экземпляр javax.microedition.lcdui.Item. Например,

```
Item display = control.initDisplayMode(control.USE_GUI_PRIMITIVE, null);
```

Для CDC реализаций поддерживающих AWT USE\_GUI\_PRIMITIVE вернёт экземпляр java.awt.Component. Для реализаций которые поддерживают и LCDUI, и AWT, требуемый тип должен быть указан вторым параметром типа String. Например

```
Item display = control.initDisplayMode(control.USE_GUI_PRIMITIVE,
    "javax.microedition.lcdui.Item")
```

Режим USE\_DIRECT\_VIDEO может использоваться только с реализациями, которые поддерживают LCDUI, а второй параметр должен быть javax.microedition.lcdui.Canvas или его потомок. Этот подход приемлем в вышеупомянутом примере. Методы videoControl могут использоваться для манипулирования размером и положением видео относительно canvas'a. Когда мы используем прямой вывод (direct video) в качестве режима дисплея, необходимо вызвать setVisible(true) чтобы видео было показано (в случае USE\_GUI\_PRIMITIVE видео показывается по умолчанию, вместе с примитивами GUI). Наконец, мы запускаем показ видео вызовом метода start().

Поскольку наш класс – потомок Canvas, мы должны реализовать метод paint(), как показано ниже

```

public void paint(Graphics g){
    g.setColor(128,128,128);
    g.fillRect(0,0,getWidth(),getHeight());
}

```

Здесь мы всего лишь заливаем холст текущим цветом фона. Затем видео отображается на Canvas посредством VideoControl.

Поскольку наш класс VideoCanvas реализует интерфейс PlayerListener, мы должны предоставить метод playerUpdate(...)

```

public void playerUpdate(Player p, String event, Object eventData) {
    if(event == PlayerListener.END_OF_MEDIA){
        if(replay == null){
            rePlay = new Command("re-play",Command.SCREEN,1);
            addCommand(rePlay);
        }
    }
}

```

В коде, показанном выше, мы просто слушаем событие END\_OF\_MEDIA и добавляем опцию повторного воспроизведения к нашим командам, когда видео подойдет к концу.

В конце давайте взглянем на метод commandAction(...) объявленный CommandListener

```

public void commandAction(
    if(c==replay){
        try{
            player.start();
        }catch(MediaException me){
            //обработка
        }
    }else if(c==close){
        player.close();
        parent.form.delete(1);
        display.setCurrent(parent.form);
        url=null;
        parent=null;
    }
}

```

Выбор опции 'Replay' попросту вызывает player.start(); Player уже инициализирован и дисплей настроен. Это всё, что требовалось. Выбор опции 'Close' закрывает видео плеер, освобождает все ресурсы, и возвращает пользователя к предыдущему экрану.

## 5 Заключение

В этом документе мы представили Mobile Media API и дали общее представление об архитектуре этого API. Затем мы объяснили его суть, предоставив детальное разъяснение средств, предоставляемых API для воспроизведения видео – листинг простейшего видео плеера. Код мидлета VideoPlayer доступен на [портале](#) Symbian Developer Network.

Во втором документе этой серии будут обсуждаться возможности Mobile Media API, включая поддержку воспроизведения звука, генерации тона и захват фото. Также более детально будет обсуждаться реализация Mobile Meida API, доступного на камерофоне Nokia 3650 под управлением Symbian OS.

## 6 Ресурсы

Исходные коды для примера VideoPlayer доступны здесь

<http://www.symbian.com/developer/downloads/java.html>

Спецификация Mobile Media API

<http://jcp.org/aboutJava/communityprocess/final/jsr135/>

Эмулятор Mobile Media API для J2ME Wiress Toolkit

<http://java.sun.com/products/mmapi/emulator/>

Series 60 Concept SDK, включая поддержку Mobile Media API

<http://forum.nokia.com>

Документы Forum Nokia

'A Brief introduction to the Mobile Media API'

'Technical Note: The Nokia 3650 Mobile Media API'