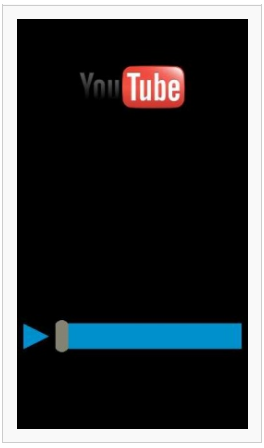
How to create a YouTube player with Java ME

This article explains how to create a YouTube player with Java ME on Nokia Belle devices. It also explains how one can easily create and attach to the video playback, a custom made control bar.

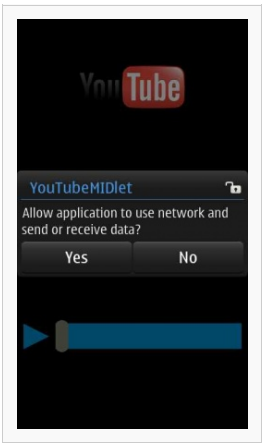
Note: The rtsp:// protocol is currently not supported on Nokia Asha Platform 1.0 or Nokia Asha Platform 1.1

Introduction

In order to be able to stream YouTube videos with Java ME, you need to first find the video's RTSP address. In this example we demonstrate how to extract and stream the following YouTube Video: Nokia Asha 311: fun, fast and always connected



No network connectivity is needed, before the play button is tapped for the first time.



Data packet transmission should be allowed by the end user.

Video playback is active.

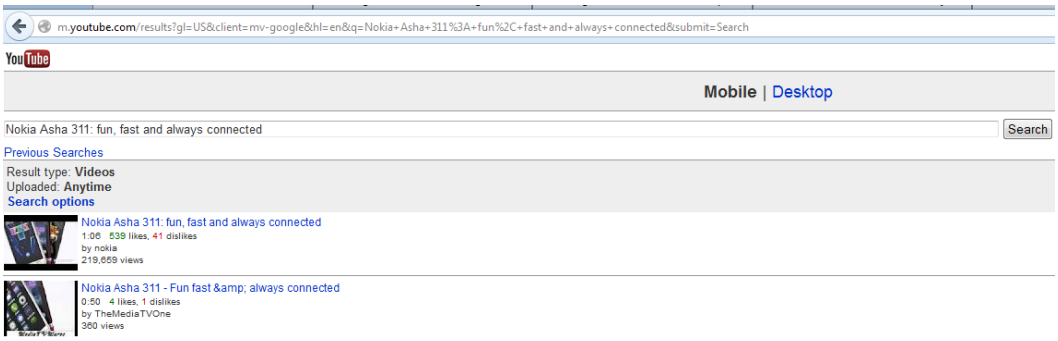
Warning: YouTube currently streams low resolution videos over RTSP, typically of size 176 x 144 pixels. This has an impact in the quality of the video on the large screen resolution supported by Nokia Belle. Trying to store the high quality streams is a violation of the terms of the service!

The following Jad attributes need to be added in the file descriptor in order to force portrait orientation (due to the low resolution videos) and in order to disable the OSK from popping up:

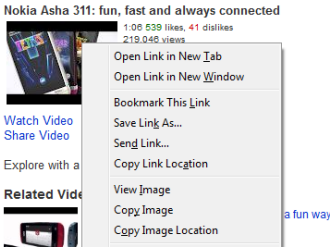
- `Nokia-MIDlet-App-Orientation: portrait`
- `Nokia-MIDlet-On-Screen-Keypad: no`

Finding the RTSP address

- You need to first open the YouTube video on your PC's browser and copy its title. In our case the title is *Nokia Asha 311: fun, fast and always connected*
- Then you need to open the mobile version of youTube on your PC's browser: `http://m.youtube.com`
- Paste the title into the search box as shown bellow and click search:



- This should return a list of search results including the video that interests you. Select it by clicking on the link. On the next page, right click on the video's thumbnail and select **Copy Link Location**.



This should copy the RTSP address for this video to the ClipBoard:

Note: **Copy Link Location** is the expected step when using the FireFox browser

- You will notice that if you hover the mouse over the thumbnail, the actual RTSP address also appears at the bottom of the browser's window:

- Finally, you will have to paste the RTSP address, you have just copied, into the code, inside the **VideoCanvas.java** file:

```
player =
```


(C) Copyright Nokia 2014. All rights reserved.

```
Manager.createPlayer("rtsp://v1.cache5.c.youtube.com/CjYLENy73wIaLQm8E_KpEOI9cxMYDSANFEIJbXYtZ29vZ2xlSARSBXdhdGNoYLm0hv_ig5HRTwwPb10/0/ni/0/0.3gp")
```

The Main MIDlet class

The streaming video can either be displayed inside a `Canvas` or `CustomItem` instance. For each case, the arguments for initializing the display mode differ. You can find more information between these differences here: Playing Video

In this article, we use a `Canvas` instance. The Main MIDlet is therefore a very simple one, where the `VideoCanvas` class is instantiated and set as the current Displayable object:

```
    protected void startApp() throws MIDletStateChangeException {
        canvas = new VideoCanvas();
        Display.getDisplay(this).setCurrent(canvas);
}
```

The supportive VideoCanvas class and the placement of the components

This is where all the components are created and painted on the `Canvas`. The `Canvas` consists of

- A YouTube logo
- The Video Window
- A Control Bar

The logo, the Video Window and the Control Bar with the Play Button, the Progress Bar and the Cursor Indicator.

Each of these components are floating inside the `Canvas`. Their final position depends on the device's screen resolution and is defined as relative to the position of the video window.

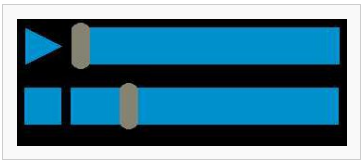
The logo is drawn at the top of the canvas with its horizontal alignment set to `HCENTER`:

```
g.drawImage(logo, width / 2, logo.getHeight() / 2, Graphics.TOP | Graphics.HCENTER);
```

The Video Window occupies the center of the Canvas, because it is set to Full Screen mode:

```
vc.setDisplayFullScreen(true);
```

The Control Bar, which consists of a Play button and a Progress Bar is placed at the lower side of the Canvas.



The Control Bar in stopped or active mode

The cursor indicator which is attached to the Progress Bar, is a filled round rectangle, painted with a light grey color.

The comments below give an overview of how the Control Bar is placed, relative to the Logo and Video Window Components

```
/*
* The leftmost X coordinate of the progress bar (barX) is defined relative to the play
button's width and the distance
* of the entire Control Bar from the sides of the screen.
*
* The control bar consists of the play button and the Progress Bar. The Control Bar's
coordinates are defined
* relative to the position of the video canvas and the size of the screen as shown
below:
*
*                                    ^
*                                    |
*                                   LOGO
*                                    |
*                                    v
*
*                                    ^
*                                    |
*                              VIDEO CANVAS
*                                    |
*                                    v
*                               CONTROL BAR
* | <- distanceBorderX -> PLAY BUTTON <- distanceFromPlayX -> PROGRESS BAR <-
```


(C) Copyright Nokia 2014. All rights reserved.

```
distanceBorderX ->|
*
*/
barX = distanceBorderX + playWidth + distanceFromPlayX;
```

As soon as we have calculated the leftmost X coordinate of the Progress Bar (barX), we can use it to calculate the width of the Progress Bar in pixels. In connection with getting the cursor's current position, we can determine the completion ratio of the playback, if we divide the distance covered by the cursor on the Progress Bar to the Progress Bar's width. The Progress Bar's width is calculated from the width of the entire screen, minus the leftmost X coordinate of the Progress Bar, minus the empty space between the Progress Bar and the right border side of the screen:

```
barWidth = width - barX - distanceBorderX;
```

The Control Bar's topmost Y coordinate is simply the southmost side of the Video Window plus a distance that we call `distanceFromVideo`:

```
//The controls are placed on the Y axis, within a safe distance from the Video canvas
controlsY = (height / 2) + (videoHeight / 2) + distanceFromVideo;
```

The distance between the Video Window and the Control Bar, is hard coded here.

The cursor extends north and south of the Control Bar, by a value that we call `cursorExtension` which denotes the number of pixels for each extension to the north and south. The cursor's north most Y coordinate can be determined as relative to the Control Bar's topmost Y Coordinate by subtracting the `cursorExtension`:

```
//The cursor extends the progress bar. The cursor's top most Y coordinate coincides with
that of the
//Control Bar but it extends north by cursorExtension pixels.
cursorY = controlsY - cursorExtension;
```

First time initialization of the Player

When the user taps on the Play button for the first time, a series of actions take place, that are not repeated twice. These include

- the creation of the `Player` instance from the RTSP address, which, in this example, is hard coded
- the retrieval of the duration of the video, which is later used as the denominator when performing division in order to translate the completion ratio of the playback to the placement of the cursor on the Progress Bar.
- Display Mode initialization of the Video Control in Full Screen.

```
//only the first time the play button is tapped
if(firstPlay){
    //The player instance is created from a hard coded rtsp address
    player =
Manager.createPlayer("rtsp://v1.cache5.c.youtube.com/CjYLENy73wIaLQm8E_KpEOI9cxMYDSANFEIJbXYtZ29vZ2xlSARSBXdhdGNoYLm0hv_ig5HRTww=/0/0/0/video.3gp");
    player.realize();
    player.prefetch();
    //The duration of the video
    duration = player.getDuration();
    //A player listener is needed so that we know, when the video has reached the
END_OF_MEDIA position
    player.addPlayerListener(this);
    //The video control instance is created and attached to this Canvas in Full SCreen
mode
    vc = (VideoControl)player.getControl("VideoControl");
    vc.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
    vc.setVisible(true);
    vc.setDisplayFullScreen(true);
    //next time the above operations will be skipped
    firstPlay = false;
}
```

Starting and stopping the video playback

Starting and stopping the playback is nothing more than listening for taps within the area where the Play or Stop Buttons are drawn. Taping is also the starting point of a drag operation. Therefore, 3 controls take place whenever a tap occurs on the screen:

- If tapping occurred within the defined rectangle area of the Cursor, then the application should drag the cursor and calculate a new Video Time for the playback. This action is irrelevant to the state of the player (active or stopped). The same state will be applied in the new playback position.

```
//If the x and y coordinates of the tapping point is within the rectangle dimensions of
the cursor, the cursor can be dragged
if((x >= cursorX) && (x <= (cursorX + cursorWidth)) && (y >= cursorY) && (y <= (cursorY
+ cursorHeight))) {
    enableDrag = true;
}
```

- If the player is stopped, and the user taps on the Play button, the video playback should resume:

```
if( isStopped && (x >= distanceBorderX) && (x <= distanceBorderX + playWidth) && (y >=
controlsY) && (y <= controlsY + controlsHeight)) {
    isStopped = false;
    repaint();
    playVideo();
}
```

- Similarly, if the video is being played and the user taps on the Stop Button, the playback should stop

```
//If the video is being played and the user taps within the square dimensions of the
```


(C) Copyright Nokia 2014. All rights reserved.

```
play/stop button, the video stops
else if( !isStopped && (x >= distanceBorderX) && (x <= distanceBorderX + playWidth) &&
(y >= controlsY) && (y <= controlsY + controlsHeight)) {
    isStopped = true;
    repaint();
    stopVideo();
}
```

Each time we resume playing, we check if the current video time equals the duration of the video. If that's the case, the we need to play the video from the beginning:

```
//if the player's current position is at the END_OF_MEDIA, the video will start playing
from 0%
if(player.getMediaTime() == duration) {
    player.setMediaTime(0);
}
```

During video playback, a thread handles the movement of the cursor along the Progress Bar

```
//A new thread handles the move of the cursor while the video progresses.
//The thread is distroyed when the video is stopped.
thread = new Thread(this);
thread.start();
player.start();
```

The code of the thread can be seen below. It simply moves the cursor in intervals of half a second by a distance calculated from the method `moveCursor()`

```
    //The thread that moves the cursor until the video playback is stopped by the user,
or the video reaches END_OF_MEDIA
    public void run() {
        while (!isStopped) {
            long currentTime = player.getMediaTime();
            moveCursor(currentTime);
            //sleep for half a second
            try {
                thread.sleep(500);
            } catch (InterruptedException e) {
                // For debugging purposes
                e.printStackTrace();
            }
        }
    }
```

Moving the cursor requires to calculate the covered distance as a multiplication of the completion playback ratio times the maximum range in pixels the cursor can move on the X axis:

```
protected void moveCursor(long currentTime) {
    //the rate of playback as a range from 0(0% completed) up to 1 (100% completed)
times the pixels of the progress bar
    double deltaX = cursorRange * (currentTime / (double) duration);
    //The new position of the cursor is the leftmost X coordinate of the progress bar
plus the above playback delta
    cursorX = barX + (int)deltaX;
    repaint();
}
```

Finally, special provision is needed, for the case when playback completes. The application needs to be notified immediately, so that the Stop button changes to Play. As we have seen above during the initialization of the Player instance, we attached to it a `PlayerListener`.

```
player.addPlayerListener(this);
```

The playerListener returns the `PlayerListener.END_OF_MEDIA` event, when playback reaches the end of the stream. This triggers a new `repaint` so that our controls remain up to date:

```
//Events received from the PlayerListener
public void playerUpdate(Player player, String event, Object eventData) {
    //If the video playback reaches END_OF_MEDIA
    if (event.equals(PlayerListener.END_OF_MEDIA)) {
        //stops the video playback
        isStopped = true;
        repaint();
    }
}
```

Considerations when porting to Series 40

When streaming video over RTSP on Series 40 devices, getting the duration of the stream using the following code is not supported:

```
player.getDuration();
```

It is therefore, advisable to disable the progress bar on Series 40 devices. This can be done by using a boolean that stores the result of the comparison of the media's duration to zero:

```
if(duration > 0) {
   isProgressDisabled = false;
```


(C) Copyright Nokia 2014. All rights reserved.

```
}
```

Depending on whether a media time stamp was successfully retrieved or not, the drawing of the progress bar and the cursor should be enabled, or disabled within the `paint` method, respectively:

```
if(!isProgressDisabled) {
   //The progress bar
   g.fillRect(barX, controlsY, barWidth, controlsHeight);

   //The cursor's color is set to a grey-ish one
   g.setColor(135, 131, 115);
   //Paints the cursor
   g.fillRoundRect(cursorX, cursorY, cursorWidth, cursorHeight, cursorWidth,
cursorWidth);
}
```

Another consideration when porting to Series 40 is that full screen video mode on Series 40 disables any canvas drawing (where the play/stop controls are added). You should therefore specify the video size and the left and top most corner of the video canvas within the screen canvas:

```
//vc.setDisplayFullScreen(true);
vc.setDisplaySize(240, 165);
vc.setDisplayLocation(0, 108);
```

Video Streaming on Series 40 devices should only be attempted over 3G or WiFi.

Note: Lack of audio playback on Nokia 311 over RTSP is a known issue. Series 40 DP1.0 touch enabled devices, do not come with 3G connectivity or WLAN and as a result they are not compatible with this example.

A version of the source code without the progress bar (modified for Series 40) can be downloaded from Media:YouTubeMIDletSourceS40.zip.

The binaries for Series 40 devices, can be found from Media:YouTubeMIDletBinariesS40.zip.

See also

- RTSP Streaming and progressive playback
- Media Time
- Java Docs - The Mobile Media API
- Multimedia in Mobile application development

(C) Copyright Nokia 2014. All rights reserved.