Media streaming on Android devices using the VideoView object available in the android.widget package. This widget allows audio or video playback local or global resources.

In this sample application the user can playback media content from an entered URL. Once the media file is loaded, the user can pause, stop, rewind or forward the playback of the file using the buttons and progress bar.

The definition of VideoView in the XML file is as follows:

```
<videoview android:id="@+id/videoView1" android:layout_width="280px"
android:layout_height="175px" android:layout_centerinparent="true"> </videoview>
```

The next step is to declaire the VideoView object in a java file:

```
private VideoView vv;
```

In the onCreate method, the reference to VideoView is created with declaration of its listeners (onErrorListener, onPreparedListener).

```
/** Called when the activity is first created.*/
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
              this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
setContentView(R.layout.main);

     //variables init
     vv = (VideoView)findViewById(R.id.videoView1);

     // listeners for VideoView:
     vv.setOnErrorListener(this);
     vv.setOnPreparedListener(this);

     url = (EditText)findViewById(R.id.mediaURL);
        url.setText("http://hermes.sprc.samsung.pl/widget/tmp/testh.3gp");
     play = (Button)findViewById(R.id.buttonPlay);
     stop = (Button)findViewById(R.id.buttonPause);
     logo = (ImageView)findViewById(R.id.imageView1);

     mediaInfo = (TextView) findViewById(R.id.mediaInfo);
     mediaTime = (TextView)findViewById(R.id.time);
     mediaTimeElapsed = (TextView)findViewById(R.id.timeElapsed);
     progress = (ProgressBar) findViewById(R.id.progressBar);
     loading = new ProgressDialog(this);
     loading.setMessage("Loading...");

     vibrator =   (Vibrator)getSystemService(Context.VIBRATOR_SERVICE);

     //flag indicating that content is ready for playback
     readyToPlay = false;
    }
```

Additionally, this code fragment also references other layout elements including EditText, Button, TextView, andProgressBar; which help to control media file playback and display additional information such as the name of the file, time elapsed, total time, or buffering progress.

Afterwards it is necessary to set source of media file. When VideoView was created then using:

```
vv.setVideoURI(uri);
```

content from URI reference is set to VideoView widget.

To properly perform playback audio or video content by VideoView and avoid problems it is good to implement two primary VideoView listeners:

OnPreparedListener - a callback which is invoked when the media file is loaded and ready to go

Registering listener:

```
vv.setOnPreparedListener(this);
```

Listener implementation:

```
         /**
         * Callback invoked when media is ready for playback
         *
         * @param mp     MediaPlayer that is ready
*/
    public void onPrepared(MediaPlayer mp) {
         Log.d(this.getClass().getName(), "prepared");
         mp.setLooping(true);
         loading.hide();

         // video size check
// (media is a video if size is defined, audio if not)
         int h = mp.getVideoHeight();
         int w = mp.getVideoWidth();
         if(h!=0 && w!=0) {
                  Log.d(this.getClass().getName(), "logo off");
                  logo.setVisibility(ImageView.INVISIBLE);
```

```
        } else {
                Log.d(this.getClass().getName(), "logo on");
                logo.setVisibility(ImageView.VISIBLE);
        }

        //onVideoSizeChangedListener declaration
        mp.setOnVideoSizeChangedListener(new OnVideoSizeChangedListener() {
                @Override
                public void onVideoSizeChanged(MediaPlayer mp, int width, int height) {
                        if(width!=0 && height!=0) {
                                Log.d(this.getClass().getName(), "logo off");
                                logo.setVisibility(ImageView.INVISIBLE);
                        } else {
                                Log.d(this.getClass().getName(), "logo on");
                                logo.setVisibility(ImageView.VISIBLE);
                        }
                }
        });


        //onBufferingUpdateListener declaration
        mp.setOnBufferingUpdateListener(new OnBufferingUpdateListener()
{
                // show updated information about the buffering progress
                @Override
                public void onBufferingUpdate(MediaPlayer mp, int percent) {
                                Log.d(this.getClass().getName(), "percent: " + percent);
                                progress.setSecondaryProgress(percent);
                }
        });

        //onSeekCompletionListener declaration
        mp.setOnSeekCompleteListener(new OnSeekCompleteListener() {

                //show current frame after changing the playback position
        @Override
                public void onSeekComplete(MediaPlayer mp) {
                        if(mp.isPlaying()) {
                                playMedia(null);
                                playMedia(play);
                        } else {
                                playMedia(null);
                                playMedia(play);
```

```
                                    playMedia(null);
                           }

mediaTimeElapsed.setText(countTime(vv.getCurrentPosition()));
                  }
         });

         mp.setOnCompletionListener(null);

         readyToPlay = true;
         int time = vv.getDuration();
         int time_elapsed = vv.getCurrentPosition();
         progress.setProgress(time_elapsed);

         timer = new CountDownTimer(time, 500) {

                  @Override
                  public void onTick(long millisUntilFinished) {

mediaTimeElapsed.setText(countTime(vv.getCurrentPosition()));
                           float a = vv.getCurrentPosition();
                           float b = vv.getDuration();
                           progress.setProgress((int)(a/b*100));
                  }

                  @Override
                           public void onFinish() {
                                    stopMedia(null);

                           }
                  };

                  //onTouchListener declaration
                  progress.setOnTouchListener(new OnTouchListener() {

                           // enables changing of the current playback position
                           @Override
                           public boolean onTouch(View v, MotionEvent event) {
                                    ProgressBar pb = (ProgressBar) v;

                                    int newPosition = (int) (100 * event.getX() /
pb.getWidth());

                                    if (newPosition > pb.getSecondaryProgress()) {
```

```
                                                newPosition = pb.getSecondaryProgress();
                                    }

                                    switch (event.getAction()) {
                                    // update position when finger is DOWN/MOVED/UP
                                    case MotionEvent.ACTION_DOWN:
                                    case MotionEvent.ACTION_MOVE:
                                    case MotionEvent.ACTION_UP:
                                                pb.setProgress(newPosition);
                                                vv.seekTo((int) newPosition *
vv.getDuration() / 100);
                                                break;
                                    }
                                    return true;
                        }
            });
```
OnErrorListener - a callback which is invoked when an error occurs during playback or setup

Registering listener:

```
vv.setOnErrorListener(this);
```
Listener implementation:

```
//onError declaration
    public boolean onError(MediaPlayer player, int what, int extra) {
              loading.hide();
              return false;
    }
```
Implementation of the onPrepared function appears complex, but it contains implementations of a only few more listeners of the MediaPlayer object. This object has more specific listeners, which provide the user with more ways to control and track media file streaming.

When onPrepared is callied it contains a reference to the MediaPlayer object on which VideoView is based on. Thanks to MediaPlayer it is possible to implement additional listeners:

•onVideoSizeChangedListener ?a callback which is invoked when the video size is known or updated

•onBufferingUpdateListener - a callback which is invoked when the status of a network stream’s buffer has changed

•onSeekCompleteListener ?a callback which is invoked when a seek operation has been completed

•onCompletionListener ?a callback which is invoked when the end of a media source has been reached during playback

Additionally, in onPrepared is using also listener of ProgressBar:

onTouchListener ?a callback which is invoked when a touch event is sent to this view. This listener is used for comfortable seeking of a media content.

Till this moment it was described how to prepare VideoView to initiate media content and related functions which helps to control it.

The next step is presentation of primary methods of VideoView object which are necessary to control media playback:

- start() ?Starts or resumes playback. If playback had previously been paused, playback will continue from where it was paused. If playback had been stopped, or never started before, playback will start from the beginning. In VideoView documentation exists function resume() which in theory should resume paused playback, but in practice it does not perform any action.
- pause() ?Pause playback.
- stopPlayback() ?Stops playback after playback has been stopped or paused. However this method is not used, because when it was called in Android 2.2 it performs file buffering even though the file was fully loaded before. In sample application this method is replaced using pause() and then seek to begin of file.
- seekTo(int milliseconds) ?Seeks to specified position expressed in milliseconds.
- getDuration() ?Gets duration of the file. Returns number of milliseconds.
- getCurrentPosition() ?Gets the current position of playback file expressed in milliseconds.
- isPlaying() ?Checks if VideoView is playing.

With the above methods the user can easily implement a simple media file player, which should be able to stream files by the insertion of a file path. It is also possible to track, for example, the current position of the playback file or percent of the file which has been correctly buffered in memory.

Issues:

- The main problem which occurs on Android 2.2 is the frequent buffering of the file even if it is fully loaded. This happens when stopPlayback() is called or when file playback is moved backward. The problem disappears when a fully loaded file plays to the end, and does not reoccur.
- The next problem which appeared during implementation is an outdated preview of the video, when playback was paused and moved backward or forward. On the preview surface the last frame played appears before the pause button was tapped. After calling the start() method, the preview updates.
- To work around this problem use the start() and pause() methods in this sequence to refresh preview of video file, when the seekTo function is done.

```
 // onSeekCompletionListener declaration
mp.setOnSeekCompleteListener(new OnSeekCompleteListener() {
        @Override
        public void onSeekComplete(MediaPlayer mp) {
                if(!mp.isPlaying()) {
                        playMedia(play);
                        playMedia(null);
                }
        mediaTimeElapsed.setText(countTime(vv.getCurrentPosition()));
}
});
```

•The third problem is no method to read metadata from media file in API Level 8. In API Level 10 there is MediaMetadataRetriever class available which helps to read that kind of data.
The only way to get metadata from media file is use external libraries which enable such functionalities.

•There is a limited number of Media Formats Supported on Android devices and it varies depending on used system version. Most up-to-date information can be found on the Android Dev Guide site:
http://developer.android.com/guide/appendix/media-formats.html
•HTTP live streaming is still not available, support for RTSP protocol (rtsp://? has not been confirmed (tried with http://m.youtube.com content)

•Only HTTP progressive streaming is confirmed to work.
Supported content on video sites for mobile phones, such as:
http://m.wp.tv
to prepare video file that will be playable from an HTTP server using progressive download one might use MP4Box or a similar tool, such as http://www.videohelp.com/tools/mp4box, to add hint tracks for RTP protocol.
Note:The video file type must be supported by a particular device.
For MP4Box, it can be done by calling below command from the command line:

```
MP4Box -hint <file_path> </file_path>
```

or

```
<installation_path>\MP4Box -hint <file_path> </file_path></installation_path>
```

and uploading the prepared file to the http server.

Ref:http://developer.samsung.com/android/technical-docs/Android-Media-Streaming-Tutorial