

UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

TELECOMMUNICATION ENGINEERING

Wang Huaming

COMPARE MULTIMEDIA FRAMEWORKS IN MOBILE PLATFORMS

Master's thesis for the degree of Master of Science in Technology submitted for
inspection in Vaasa, 1st of April, 2012.

Supervisor D.Sc. (Tech.) Mohammed Salem Elmusrati

Instructor M.Sc. Tobias Glocker

TABLE OF CONTENTS

ABBREVIATIONS	5
1. INTRODUCTION	10
1.1. Multimedia software revolution.....	10
1.2. Multimedia underlying technology.....	12
1.2.1. Memory technology.....	12
1.2.2. Silicon technology	13
1.2.3. Display technology	13
1.2.4. Camera technology	14
1.2.5. Video compression	15
1.2.6. Audio compression.....	16
1.3. Organization of the thesis	17
2. BASIC OF MULTIMEDIA	18
2.1. Audio	18
2.2. Video.....	19
2.3. Encoder	21
2.4. Decoder.....	22
2.5. Container Format	22

2.6.	Transmission Protocols	23
2.6.1.	RTP	23
2.6.2.	RTSP	24
2.6.3.	MMS	25
2.6.4.	HTTP Progressive Download	25
3.	MULTIMEDIA FRAMEWORK OVERVIEW	27
3.1.	Open Source GStreamer	28
3.2.	Android Stagefright	29
3.3.	Microsoft Silverlight Media Framework	30
4.	COMPARSION OF MULTIMEDIA FRAMEWORKS	32
4.1.	Design Architecture	32
4.1.1.	GNOME GStreamer	32
4.1.2.	Android Stagefright	34
4.1.3.	Microsoft Silverlight Media Framework	37
4.2.	License and Legal	38
4.3.	Developer Support	39
4.4.	Implementation Language and Language binding	40
4.5.	Supported User Cases	40

4.5.1. GStreamer simple media player	41
4.5.2. SMF simple media player.....	42
4.5.3. Stagefright simple media player	46
4.6 Performance	49
5. CONCLUSION	50
REFERENCE	53
APPENDIX I.....	55
APPENDIX II.....	82
APPENDIX III	94

ABBREVIATIONS

AAC	Advance Audio Coding
AMR-NB	Adaptive Multiple Rate Narrow Band
AMR-WB	Adaptive Multiple Rate Wide Band
API	Application Program Interface
ARM	Advanced RISC Machine
ASCII	American Standard Code for Information Interchange
AVC	Advance Video Codec
CD	Compact Disc
CRT	Cathode Ray Tube
GNU	GNU's Not Unix
GPL	General Public License
HD	High Definition
HE-AAC	High Efficiency Advance Audio Coding
HTTP	Hyper Text Transfer Protocol
HW	Hard Ware
IDE	Integrated Development Environment
IIS	International Information Server
IP	Internet Protocol
JNI	Java Native Interface

LGPL	Lesser General Public License
MB	Mega Byte
MEF	Managed Extensibility Framework
MMS	Microsoft Media Server
MPEG	Motion Picture Expert Group
OMX	OpenMAX
PCM	Pulse Code Modulation
QOS	Quality of Service
RGB	Red Green Blue
RTP	Real-time Transfer Protocol
RTSP	Real Time Streaming Protocol
SDK	Software Development Kit
SDP	Session Description Protocol
SMF	Silverlight Media Framework
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URL	Universal Resource Locator
XML	Extensible Markup Language
VCD	Video CD

DVD	Digital Versatile Disc
HDTV	High Definition TV
RAM	Random Access Memory
NOR	Not OR
NAND	Not AND
XIP	Execute In Place
RV	Real Video
WMV	Windows Media Video
QVGA	Quarter Video Graphic Array
WVGA	Wide Video Graphic Array
DSC	Digital Still Camera
CPU	Common Processor Unit
CMOS	Complementary Metal Oxide Semiconductor
CCD	Charge Coupled Device
3GPP	Third Generation Partnership Project
ISO	International Standard Organization
UMTS	Universal Mobile Telecommunications System
I-TUT	ITU Telecommunication Standardization Sector
JVT	Joint Video Team
RA	Real Audio

WMA	Windows Media Audio
DLS	Downloadable Sound
PPI	Pixel Per Inch
LCD	Liquid Crystal Display
OLED	Organic Light Emitting Diode
DRAM	Dynamic Random Access Memory
SRAM	Static Random Access Memory

UNIVERSITY OF VAASA**Faculty of technology****Author:**

Huaming Wang

Topic of the Thesis:

Compare Multimedia Frameworks in Mobile Platforms

Supervisor:

Mohammed Salem Elmusrati

Instructor:

Tobias Glocker

Degree:

Master of Science in Technology

Degree Programme:

Degree Program in Information Technology

Major of Subject:

Telecommunication Engineering

Year of Entering the University:

2004

Year of Completing the Thesis:

2012

Pages: 112

ABSTRACT: Multimedia feature is currently one of the most important features in mobile devices. Many modern mobile platforms use a centralized software stack to handle multimedia requirements that software stack is called multimedia framework. Multimedia framework belongs to the middleware layer of mobile operating system. It can be considered as a bridge that connects mobile operating system kernel, hardware drivers with UI applications. It supplies high level APIs that offers simple and easy solutions for complicated multimedia tasks to UI application developers. Multimedia Framework also manages and utilizes low lever system software and hardware in an efficient manner. It offers a centralize solution between high level demands and low level system resources.

In this M.Sc. thesis project we have studied, analyzed and compared open source GStreamer, Android Stagefright and Microsoft Silverlight Media Framework from several perspectives. Some of the comparison perspectives are architecture, supported use cases, extensibility, implementation language and program language support (bindings), developer support, and legal status aspects. One of the main contributions of this thesis work is that clarifying in details the strength and weaknesses of each framework. Furthermore, the thesis should serve decision-making guidance when on needs to select a multimedia framework for a project.

Moreover, and to enhance the impression with the three multimedia frameworks, a basic media player implementation is demonstrated with source code in the thesis.

KEYWORDS: Multimedia Framework, GStreamer, Stagefright, Silverlight Media Framework

1. INTRODUCTION

Mobile devices like handset and internet tablet have widely involved in people's life. By fast revolution of electronic components mobile device gained more and more power to handle multimedia data. Hence multimedia experience becomes one of the most important performance indexes and selling point for these devices. People not only watch HD movies, listen music, edit captured video within their handsets but also make a VoIP call, join a video conference, listen the Internet radio, share media content with home network devices, listen their favorite music tracks on car entertainment system as well as watch on-line video streams over the network. To achieve a great multimedia experience and find a centralized and efficient solution to support all above fantastic features and functions almost all the modern mobile platforms use a Multimedia Framework to handle multimedia requirements.

A Multimedia Framework is a software framework that handles media data on a computer device and through a network. A good multimedia framework offers intuitive APIs and modular architecture to easily add support for new audio, video and container formats and transmission protocols. It is meant to be used by applications such as media player and audio or video editor, but can also use to build video conferencing applications, media converters and other multimedia tools.

There are several popular multimedia frameworks in modern operating systems, like GStreamer, a cross-platform open source multimedia framework, primary used in Linux operating system. Nokia and Intel successfully use it in MeeGo mobile platform. Silverlight Media Framework, a multimedia framework produced by Microsoft originally is used in desktop environment and now it also offers service to Windows Phone 7 platform. Stagefright, a multimedia framework introduced by Google for Android 2.0 onwards mobile devices to replace previous OpenCORE media framework.

1.1. Multimedia software revolution

During the past couple of years, we have witnessed a rapid increase in the multimedia capabilities of mobile devices. The market has shown that the mobile device is a natural multi-functional device, which in all likelihood will serve as disruptive technology for

many traditional, portable consumer electronic devices, such as digital cameras, music and video players. Current trends in technology will provide solutions for pushing multimedia capabilities forward at the same rapid pace for another four or five years.

Multimedia software's two considerations when designing a new generation of mobile platforms are market trend and technology evolution. The challenge of design multimedia software is to find the right balance between cost, functionality and performance, flexibility, and time to market. Every trade-off affects methods development and choice of hardware, software, and tools. (Jim & Fredrik 2004: 1).

In 2001 Nokia launched first 2.5G phone Nokia 7650 which ran with Symbian OS and in 2007 Symbian released version 7.0s which included a completely new multimedia framework. The MMF provides a light weight, multi-threaded framework for handling multimedia data. Symbian MMF supports audio and video playback, recording and streaming, and also image related functions.

Fourth quarter of 2005 Nokia launched the first Maemo based device Nokia 770. And Maemo is a Linux kernel 2.6 based mobile platform use GStreamer as MMF. Nokia launched couple of Maemo successor devices like N800, N810, N810 WiMAX, N900, and the last one is N9.

In 2007 November 5, Open Handset Alliance unveiled their first product Android, a mobile device platform built on the Linux kernel version 2.6. The MMF of Android is based on PacketVideo's OpenCORE. OpenCORE supports playback and recording of many popular audio and video formats and as well as static image files. In November 2009, Android 2.0 was released. The release included a new designed and simplified MMF called Stagefright and Android totally dropped OpenCORE support from Android 2.2 afterwards.

In 2007 January 9, Steve Jobs CEO of Apple unveiled the first iPhone and the 5th generation iPhone was announced on October 4, 2011. iPhone devices are running with Apple iOS mobile operating system and iOS uses couple of multimedia frameworks to deal with multimedia tasks. Media Player framework is to play songs, videos, audio books, or audio podcasts from user's iPod library. AV Foundation framework is to play and record audio using a simple Object-C interface. Audio Toolbox framework is to

play audio with synchronization capabilities, access packets of incoming audio, parse audio streams, and convert audio formats and record audio with access to individual packets. Audio Unit framework is to connect to and use audio processing plugins. OpenAL framework is to provide positional audio playback in games and other applications. (iOS Developer Library: About Audio and Video).

In 2010 Microsoft announced their new mobile platform Windows Phone 7 also known as Windows Phone Mango release. Microsoft Silverlight media framework is used to handle media local player back and streaming tasks in Windows Phone 7. In 2012 Microsoft will publish Windows Phone 8 Apollo release. There are going to have completely fresh design multimedia framework to replace the Silverlight media framework. People expect the new MMF can bring more rich multimedia experience to end user.

1.2. Multimedia underlying technology

The multimedia software revolution has never happened alone. Silicon technology, memory technology, camera technology, display technology, video and audio compression algorithm, mobile operating system and the most important marketing needs are the underline driving force for its revolution.

1.2.1. Memory technology

Two main types of memory are found in mobile devices: non-volatile program and data storage, and fast-access random access memory (RAM), for active software. By tradition, not-or (NOR) flash memory has been used for non-volatile storage. Although this type of memory is slow to update, it is quickly read, which facilitates execute-in-place (XIP) functionality. In other words, the phone's processor can fetch code directly from memory for execution. More and more manufacturers replace NOR flash memory with not-and (NAND) flash memory, which is denser and yields substantially greater capacity from the same silicon area. In addition, NAND flash memory has a substantially lower cost per megabyte—typically one-third to one-sixth that of NOR flash memory. But because the random access time associated with NAND flash

memory is quite long, it is not suitable for XIP functionality. Instead, it is more characteristic of secondary storage, much like a hard disk in a PC.

Present-generation mobile phones also have substantially more RAM than their predecessors. A primary reason for this is that users are being allowed to handle and generate multimedia content in their devices. Manufacturers are also moving away from static RAM (SRAM) to dynamic RAM (DRAM), which is substantially denser.

1.2.2. Silicon technology

Advancements in silicon technology and processor architecture are opening the way for vastly improved CPU performance. Advances in the area of silicon technology continue to follow Moore's Law. The International Technology Roadmap for Semiconductors reported that the silicon geometry of CPUs and ASICs entering into production in 1998 was 250nm; in 2000, it had shrunk to 180nm; in 2002, 130nm; in 2004, 90nm; and the projected geometry in 2007 is 65nm. An additional benefit of smaller geometries is faster clock frequencies. In general, greater transistor density means greater potential for more advanced and powerful processors. There are several ways of increasing the processing performance of CPUs. Longer pipelines yield higher clock frequency, and more advanced instructions, such as DSP-like extensions (for example, the ARM9E family of processors) increase the ability to perform several operations per clock cycle (for example, multimedia extensions of the ARM11 family). Because external memory and bus structures cannot keep up with increases in CPU speeds, more advanced cache, buffer memory, and branch predictions are used to increase effective application performance.

1.2.3. Display technology

Large and bright color displays have become a strong selling point for mobile phones. Display technology is evolving rapidly. The QVGA display introduced in phones in 2003 and now WVGA display became common in mainstream smart phones.

The pixel density of displays in mobile phones is higher than that of displays in laptop or desktop PCs. Laptops have some 100-135 pixels per inch (PPI), whereas high-end

mobile phones have between 150 and 300PPI. These displays will have high visual quality; graphics will appear very sharp, but most people will still be able to discern individual pixels. The resolution limit of the human eye is approximately 0.5 minutes of arc which corresponds to about 700 PPI at a viewing distance of 25cm. Good printers easily exceed this resolution, which is why most people prefer reading and viewing printed text and images.

Where power efficiency is concerned, the majority of dominating LCD systems leaves much to be desired. Most present-day LCD systems consist of a TFT panel on top of a backlight panel. The polarizer and color filters shutter and modulate the backlight. This method of producing an image is highly inefficient, however. In fact, more than 90% of the backlight intensity is lost. Organic light emitting diodes (OLED) consists of electro-luminescent pixel elements that emit light directly. Apart from lower overall power consumption, this technology offers greater brightness and contrast and faster response times than TFT displays. OLED display technology currently has only a small-scale presence in the market due to issues with aging, manufacturing yields and cost.

1.2.4. Camera technology

Today built-in camera is a must-have feature of smart phones and real-time processing of megapixel resolution images is demanding and often requires hardware acceleration to assist in image compression/decompression, color space conversion, scaling and filtering. As costs continue to fall, many of the standard features associated with dedicated digital still cameras (DSC) are already in mainstream mobile phones—for example, multi-megapixel sensors, flash, autofocus, optical zoom and face recognition.

Two image-sensor technologies currently dominate: complementary metal oxide semiconductor (CMOS), and charge coupled device (CCD). Compared to CMOS, CCD technology generally offers better sensitivity and signal-to-noise levels, but it is also more expensive and power-hungry. This technology has mainly been reserved for high-end megapixel camera phones. CMOS technology has been more common in sub-megapixel cameras (such as popular 0.3 megapixel VGA front end cameras). In terms of

resolution and sensitivity, however, it is fast approaching CCD, and many new CMOS based multi-megapixel camera phones will be introduced in coming years.

1.2.5. Video compression

The Third-generation Partnership Project (3GPP) stipulates which codecs may be used for services in UMTS: H.263 is mandatory; MPEG-4 and H.264 are optional. Because a good deal of available content has been coded in Real Video (RV) and Windows Media Video (WMV), support is also being considered for these proprietary formats, in particular, where viewing or browsing is concerned.

Historically, two organizations have contributed toward the standardization of video codecs. The ISO Moving Pictures Expert Group developed MPEG-1, MPEG-2 and MPEG-4, which is used for VCD, DVD and HDTV. ITU-T developed H.261 and H.263, mainly for video-conferencing. In 2001, the two organizations formed the Joint Video Team (JVT) to develop a new recommendation or international standard targeting greater compression efficiency. In 2003, JVT announced ITU-T H.264 and Advanced Video Coding (AVC), Part 10 in the MPEG-4 suite.

More efficient compression (thanks to the H.264 codec) will improve perceived video quality, especially video telephony and streaming video at bit rates as low as 64kbps. However, these gains in compression efficiency are not free. They call for a considerable increase in computational complexity and memory, which adds to the overall cost and energy consumption of the phone. Because decoding has less effect on performance than encoding, and because the ability to consume emerging content is a top priority, but by the improvement of memory and processor in performance and price the high-end mobile device in market all support H.264 encoding and decoding.

VP8 is a video codec and originally was owned and released by On2 on September 13, 2008 for replacing its predecessor VP7. In 2010 Google acquired On2 and released VP8 source code with public license that makes VP8 free software. Google is gradually transcoding all YouTube clips with VP8 to replace the Adobe Flash Player and H.264. On May 19, 2010, the WebM project was launched. Mozilla, Opera, Google and more than forty other publisher, software and hardware vendors contributed a major effort to

adopt VP8 as video format in the WebM container for HTML5. So we have reason to believe that VP8 is going to be next popular video codec when HTML5 gain big population in Internet.

1.2.6. Audio compression

More is the operative word in current audio trends. More codec formats, more synthetic audio formats, more audio effects, and more simultaneous audio components. New use cases and competing codecs are behind the drive for more audio codecs. At present, the most common audio codecs are MP3, AAC, Vorbis and WMA. The trend in audio codecs is for greater support of low bit rates. At the same time, voice codecs, such as AMR-WB, are evolving to provide support for general audio at bit rates that are economically reasonable for streaming and messaging. On the other hand we see several products that aim at high quality audio playback (N9 with Dolby, HTC with HQ headsets). There companies try to support high bitrates, lossless audio codec (like FLAC) and audio enhancement (Dolby, SRS).

A new format for synthetic polyphonic audio is mobile downloadable sound (DLS), which allows users to customize synthesized sound. For example, with DLS, users could add extra instruments with a sound that is specific to, or characteristic of, a given melody.

The current generation of mobile phones uses audio effects, such as equalizers and dynamic range compression. These effects alter the frequency spectrum to adapt to output devices (small speakers and headsets) and compress peaks in volume (with a loss in dynamic resolution). New effects being introduced include chorus, reverberation, surround sound and positional three-dimensional audio. As its name implies, the chorus effect makes one sound signal sound like multiple sound signals. Likewise, the reverberation effect imitates the reflection of sounds off walls.

Most current-generation phones support stereo headsets. The surround-sound effect has been introduced to enhance the stereo listening experience. The positional three-dimensional audio effect makes it possible to move sound sources in a virtual three-

dimensional space so that listeners perceive sound sources as if they are coming from a specific direction and distance.

1.3. Organization of the thesis

Chapter 2 present relevant multimedia fundamental components, introducing digital audio, digital video, encoder, decoder, media formats and common multimedia streaming protocols.

Chapter 3 gives an overview for three popular multimedia frameworks, introducing open source GStreamer, Android Stagefright and Windows Phone 7 Silverlight media framework.

Chapter 4 analyzes the three frameworks from different aspects which are architecture, license and legal, user support, implementation language and language binding, and real user cases.

Chapter 5 presents a discussion of the results achieved in chapter 4 and the conclusions. Finally, the challenges faced during this master's thesis project are summarized.

2. BASIC OF MULTIMEDIA

2.1. Audio

Sound is a sequence of waves of pressure that propagates through compressible media such as air or water. The behavior of sound propagation is generally affected by three factors: a relation between density and pressure. The relationship affected by temperature, determines the speed of sound within the media. The propagation is also affected by the motion of the media itself, for example sound moving through wind. Independent of the motion of sound through the media if the media is moving the sound is further transported. The viscosity of the media also affects the motion of sound wave. It determines the rate at which sound is attenuated.

The perception of sound in any organism is limited to a certain range of frequencies. For humans, hearing is normally limited for frequencies between about 20 Hz and 20 kHz and the upper limit generally decreases with age.

An audio signal is a one-dimensional function, a single value varying over time. It is continuous in both value and time; that is at any given time it can have any real value and there is a smoothly varying value at every point in time. No matter how much we zoom in there are no discontinuities, no singularities, no instantaneous steps or points where the signal ceases to exist.

Audio had always been manipulated as an analog signal but for storing it, copying it, processing it and transmitting it in consumer electronic device we need to transform the analog signal to digital signal. Sampling theorem is published by Claude Shannon in 1949 states that not only we can go back and forth between analog and digital signal but also lays down a set of condition for which conversion is lossless and the two representations become equivalent and interchangeable. Pulse Code Modulation is a most common representation for raw audio. PCM encoding can be characterized in three parameters.

The first parameter is the sample rate. The highest frequency an encoding can represent is called Nyquist Frequency. The Nyquist Frequency of PCM happens to be exactly half the sampling rate. Therefore the sampling rate directly determines the highest possible

frequency in the digitized signal. For example analog telephone system traditionally band-limited voice channels to just less than 4 kHz, so digital telephony and most classic voice application use an 8 kHz sample rate. As power, memory and storage increased, consumer electronic device now can offer 11, 16 and then 22 and even 32 kHz sampling rate with each increase sampling rate and the Nyquist Frequency it is obvious that the high frequency end becomes a little clearer and the sound more natural.

The second fundamental PCM parameter is the sample format which is the format of each digital number. Early PCM was eight-bit linear, encoded as unsigned byte. That means each sampling point value can be represented as an eight-bit unsigned binary number. There are totally 256 levels to represent sampling point value. Digital telephone typically uses one of two related non-linear eight bit encoding called A-law and Mu-law. These formats encode roughly 14 bit dynamic range into eight bits by spacing the higher amplitude values farther apart. A-law and Mu-law obviously improve quantization noise compare to linear eight bits and voice harmonics especially hide the remaining quantization noise well. Currently using 16 bit signed linear values is most common in mobile devices.

The third PCM parameter is the number of channels. The convention in raw PCM is to encode multiple channels by interleaving the samples of each channel together into a single stream.

2.2. Video

Video is a sequence of still images which are for reconstruction representing scenes in motion. Video can also be recording, digitalizing, processing, storing, and transmitting over network. Video is like audio but with two additional spatial dimensions, X and Y. The Sampling Theorem applies to all three video dimensions just as it does to the single time dimension of audio.

Audio and video are obviously quite different in practice. Like compare to audio, video is huge. Raw CD audio is about $44100(\text{samples per second}) \times 16(\text{bits per sample}) \times 2(\text{channels, left and right}) = 1.4$ megabits per second and raw 1080i HD video is over 700 megabits per second.

The most obvious raw video parameters are the width and height of the picture in pixels. Standards have generally specified that digitally sampled video should reflect the real resolution of the original analog source so large number of digital video also uses non-square pixels.

The second video parameter is the frame rate which is the number of full frames per second. The higher frame rate the smoother the motion that can bring to consumer. Back to the early days of broadcast video engineers sought the fastest practical frame rate to smooth motion and to minimize flicker on phosphor-based CRTs. The solution is to interlace the video where the even lines are send in one pass and the odd lines in the next. Each pass is called a field and two fields sort of produce one complete frame. In a 60 field per second picture, the source frame rate is actually 60 full frames per second and half of each frame, every other line is discarded. That is why we cannot de-interlace a video simply by combining two fields into one frame because they are actually not from one frame.

The human eye has three apparent color channels: red, green, and blue. Most displays use these three colors as additive primaries to produce a full range of color output. Video can be and sometimes is represented with red, green, and blue color channels but RGB video is atypical. The human eye is far more sensitive to luminosity than its color and RGB tends to spread the energy of an image across all three color channels. Because those reasons and also television happened to start out black and white video usually is represented as a high resolution luma channel and lower resolution chroma channels. The luma channel Y is produced by weighting and then adding the separate red, green and blue signals. The chroma channels U and V are then produced by subtracting the luma signal from blue and the luma signal from red.

When YUV is scaled, offset, and quantized for digital video, it is usually more correctly called YCbCr but the generic term YUV is widely used to describe all the analog and digital variant of this color model.

The U and V chroma channels can have the same resolution as the Y channel, but because the human eye has far less spatial color resolution than spatial luminosity resolution, chroma resolution is usually halved or even quartered in the horizontal

direction, the vertical direction, or both, usually without any significant impact on the apparent raw image quality. Practically every possible subsampling variant has been used at one time or another, but the common choices today are 4:4:4 video which is not actually subsampled at all. 4:2:2 video in which the horizontal resolution of the U and V channels is halved and most common of all 4:2:0 video in which both the horizontal and vertical resolution of the chroma channels are halved, resulting in U and V planes that are each one quarter the size of Y.

In audio we always represent multiple channels in a PCM stream by interleaving the samples from each channel in order. Video use both packed formats that interleave the color channels, as well as planar formats that keep the pixels from each channel together in separate planes stacked in order in the frame. There are at least 50 different formats in these two broad categories with possibly 10 or 15 in common use.

Each chroma subsampling and different bit-depth requires a different packing arrangement and so a different pixel format. For a given unique subsampling there are usually also several equivalent formats that consist of trivial channel order rearrangement or repacking, due either to convenience once-upon-a-time on some particular piece of hardware or sometimes just good old-fashion spite. Pixels formats are described by a unique name or *fourcc* code. Be aware that *fourcc* codes for raw video specify the pixel arrangement and chroma subsampling but generally do not imply anything certain about chroma siting or color space. (A Digital Media Primer for Geeks: 2).

2.3. Encoder

After the audio wave or video image is captured normally it will be processed before it is saved into electronic device. The reason for that is reducing the storage space and bandwidth required for transmission of stored audio information. Let's look at simple example; we have a normal one minute duration CD music track. The file size of it will be $44100 \text{ samples/sec} \times 16 \text{ bits/sample} \times 2 \text{ channels} \times 60 \text{ sec} = 84.6 \text{ Mb}$. When we download that over internet with an old 56kbit modem typically connected at 44kbit, it will take $84.6 \text{ Mb} / 44000 \text{ baud} = 1924 \text{ seconds} \sim 32 \text{ minutes}$. And that same clip if it is

encoded with MP3 encoder then the size will be shrunked to 1-3 MB. Now a day the bandwidth of modern wireless connection can provide 56 Mb/sec so the audio streaming is well supported from network side.

2.4. Decoder

The decoder reverses the process of encoder. It decompresses encoded audio and video data producing a raw data stream which can feed to output hardware like graphic card or sound card.

2.5. Container Format

Chunks of raw audio and video data have no externally-visible structure, but they are often uniformly sized. We could just string them together in a rigid predetermined ordering for streaming and storage. Compressed frames are not necessarily a predictable size and we usually want some flexibility in using a range of different data types in streams. If we string random formless data together we lose the boundaries that separate frames and do not necessarily know what data belongs to which streams. A stream needs some generalized structure to be generally useful.

In addition to our signal data we also have PCM and video parameters. There is plenty of other metadata we also want to deal with, like audio tags, lyrics, video chapters and subtitles, all essential components of rich media. It makes sense to place this metadata within the media itself.

Structuring formless data and disparate metadata is the job of container. Container provide framing for the data blobs, interleave and identify multiple data streams, provide timing information and store the metadata necessary to parse, navigate, manipulate and present media.

- MP4 (.mp4) is a popular container format defined in the MPEG-4 Part 14 standard. It supports almost any kind of media data. Typically an MP4 container contains video and audio streams encoded with H.264 and AAC, respectively.

- 3GP (.3gp) is widely utilized on 3G mobile phones. 3GP is defined as an extension of MPEG-4 Part 12. 3GP normally stores video streams encoded with MPEG-4 Part 2, H.263, or H.264 and audio streams with either AAC (LC profile) or HE-AAC.
- Ogg (.ogg) is an open container format developed by the Xiph.Org Foundation. Ogg generally combines the Theora and Vorbis codecs.
- Matroska (.mkv) is an open source and open standard container format. It can hold unlimited number of video, audio, picture or subtitle tracks in one file and it intend to serve as a universal format for storing common multimedia content. (Matroska).
- WebM (.webm) is an audio-video format designed for using with HTML5 video. It use Matroska as baseline and combines the VP8 video and Vorbis audio. (WebM an open web media project).

2.6. Transmission Protocols

2.6.1. RTP

The Real-time Transmission Protocol provides the low-level transport functions suitable for applications transmitting real-time data, such as video or audio, over multicast or unicast services. RTP works over IP networks and thus it is compatible with various transmission channels. The RTP standard consists of two elementary services, transmitted over two different channels. One of them is the real-time transport protocol which carries the data and the other works as control and monitor channel named RTP control protocol.

RTP packets are encapsulated within UDP datagrams. This step incorporates a high throughput and efficient bandwidth usage. On the other hand there is no quality of service (QoS) features in IP version 4 to ensure a constant bandwidth. Moreover packet losses in UDP transmissions need to be tracked and handled by the application layer. Therefore RTP packets contain timestamps and sequencing information within their header.

The primary function of RTCP is to provide feedback on the quality of the data distribution. The feedback may be directly useful for control of adaptive encodings

along with fault diagnostics in the transmission. A further RTP option is point to multipoint delivery, known as multicast.

2.6.2. RTSP

The Real-Time Streaming Protocol is a client-server signaling scheme based on messaging in ASCII-form. It establishes and controls either a single or several time-synchronized streams of continuous media such as audio and video. The protocol is intentionally similar in syntax and operation to HTTP and therefore employs the option of using proxies, tunnels and caches. RTSP works well both for large audiences as well as single-viewer media-on demand.

In contrast to SIP an RTSP session is unidirectional, a media server may either play or record data, with the direction indicated at stream setup time.

Since the amount of data transmitted within an RTSP session is low and mostly non time critical, a TCP channel is used by default. RTSP itself does not deliver data, though the RTSP connection may be used to tunnel RTP traffic.

RTSP provides control functionality such as pause, fast forward, reverse and absolute positioning and works very similar to a video recorder remote control.

The necessary supplementary information in the negotiation phase is carried within the Session Description Protocol which is transmitted as attachment of the appropriate RTSP response.

SDP includes the session name information how to receive the media in terms of addresses, ports and transport protocols. Furthermore extensive information on the media is provided, format (MPEG-4 video, AAC/CELP audio, etc.), timing and initial decoder configuration. SDP is based on a compact encoding scheme. A single character denotes the type followed by a “=” character. The desired value is directly attached to the corresponding type line.

2.6.3. MMS

Microsoft Media Server is the name of Microsoft's proprietary network streaming protocol used to transfer unicast data in Windows Media Services (previously called NetShow Services). MMS can be transported via UDP or TCP. The MMS default port is UDP/TCP 1755.

Microsoft deprecated MMS in favor of RTSP (TCP/UDP port 554) in 2003 with the release of the Windows Media Services 9 Series, but continued to support the MMS for some time in the interest of backwards compatibility. Support for the protocol was finally dropped in Windows Media Services 2008.

Note however that Microsoft still recommends using "*mms://*" as a "protocol rollover URL". As part of protocol rollover a Windows Media Player version 9, 10, or 11 client opening an "*mms://*" URL will attempt to connect first with RTSP over UDP and if that fails it will attempt RTSP over TCP. After an RTSP attempt fails, Windows Media Player versions 9 & 10 will attempt MMS over UDP, then MMS over TCP. If using Windows Media Player 11 and an RTSP attempt fails, or if using a previous version of Windows Media Player and MMS fails, a modified version of a HTTP over TCP connection will be attempted. This modified version is referred to by some third parties as MMSH and by Microsoft as MS-WMSP (Windows Media HTTP Streaming Protocol). The URI scheme "*mms://*" has also been proposed to be used for the unrelated Multimedia Messaging Service (MMS) protocol.

2.6.4. HTTP Progressive Download

Progressive download typically can be realized using a regular HTTP server. Users request multimedia content which is downloaded progressively into a local buffer. As soon as there is sufficient data the media starts to play. If the playback rate exceeds the download rate, then playback is delayed until more data is downloaded.

The Hypertext Transfer Protocol is widely known as the protocol of the World Wide Web. It is connection-based: a client opens a connection with a server. It can then send requests, and the server responds to these requests. When the connection is closed, no

more requests can be sent by the client, and no more responses can be sent by the server.

Usually, HTTP uses TCP to transport data, as packet loss could result in incomplete web pages. However, with some effort to overcome data loss, especially when establishing connections, it should be able to use UDP as well.

Headers are used to describe both the HTTP packet (e.g., whether it's a request or a response) and its contents, if present. They are supplied in plain-text inside an HTTP packet, in the form key: value, where headers may appear in any order. The HTTP specification defines many headers that may or may not be used. New headers may be defined and used as well (although then the protocol might not be recognized by other HTTP implementations such as browsers or web servers, anymore). This approach does introduce some overhead when compared with protocols with fixed headers, such as RTP. Since HTTP's headers are in plain-text, human-readable form, they include the name of the header (for fixed headers, it is not necessary to include a header's name, since it is identified by its position within the header), and no effort is made to compress them. However, this does not need be problematic when the number of headers is small or the payload is large.

The specification of HTTP1.1 provides two types of transfer encoding: a default one, which sends a payload in one large HTTP packet (which may consist of more than one TCP packet), and 'chunked', which splits the payload into a sequence of separate HTTP packets that are then sent consecutively. HTTP1.1 requires that the length of the payload be included as a header field inside an HTTP packet. However when the transfer encoding is set to chunked, only the length of a chunk needs to be included, not the length of the entire stream. This makes live streaming easier, since the length of a live stream is unknown beforehand. Nevertheless, the Shoutcast internet radio protocol uses HTTP without using a chunked encoding.

3. MULTIMEDIA FRAMEWORK OVERVIEW

A multimedia framework is a software package composed of a set of software libraries. The primary function of such a framework is to handle media objects, like introduced in previous chapter. A good multimedia framework offers an intuitive API and a plug-in architecture. Thereby, new codecs, new formats, new capture devices, new communication procedures, etc. can comfortably be added. Its main purpose is to be integrated into multimedia applications such as multimedia players, multimedia servers or multimedia editing software. Furthermore, the specialized libraries and interfaces of a multimedia framework make it possible to combine new and customized multimedia solutions. Multimedia frameworks process media with various handlers for formats, streams and contents. They are equipped with fresh codecs, formats, multiplexers, readers, writers, etc. Additionally, frameworks try to automate the media handling by setting up appropriate media processing queues. A modern multimedia framework ought to be extensible because of the rapid development concerning all parts of multimedia processing.

In my thesis project three popular multimedia frameworks will be studied, analyzed and illustrated here. Open source GStreamer is an over 10 years open source project. It is mostly known for its use in media players although the APIs and the plugin collection have much offer for audio video composing and editing application as well. Although GStreamer originally is used for Linux environment, it has been port to MeeGo platform an ARM based mobile Linux operation system. It ran inside internet tablet Nokia 770, N800, N810, N810 WiMAX, N900 and the latest and last product is Nokia N9. Google Android Stagefright is newly designed media framework which introduced in Android 2.0 afterwards release and it is for replacing OpenCORE media framework. The Stagefright source code can be downloaded from Google repository that enables to HW manufactures and developers to implement and integrate new multimedia features and codecs to Stagefright. Microsoft Silverlight Media Framework is used to handle Windows based desktop application. Microsoft redesigns and optimizes it recently to make it also suitable for developing media playback application in Microsoft Windows Phone platform.

3.1. Open Source GStreamer

GStreamer is a flexible and mature open source multimedia framework for multimedia applications. It has a pipeline design to compose processing chains. The pipeline design was chosen to decrease the queues overhead. This supports applications with high demands on latency.

The framework contains plug-ins that provides various libraries and tools. Plug-ins are considered parsers, codecs, sinks, multiplexers and demultiplexers. The pipeline structure defines the flow of the media data. The project has been split into modules. The GStreamer core module is media agnostic. It provides a library of functions which define a framework for plug-ins, data flow and media type handling. Specific media such as audio, video and text are introduced in GStreamer plug-ins modules. A documented API and sample applications are available. A number of applications use the GStreamer framework including applications developed for the market of mobile communication. The GStreamer project is released under the LGPL and is open for development and free for download. (GStreamer Application Development Manual).

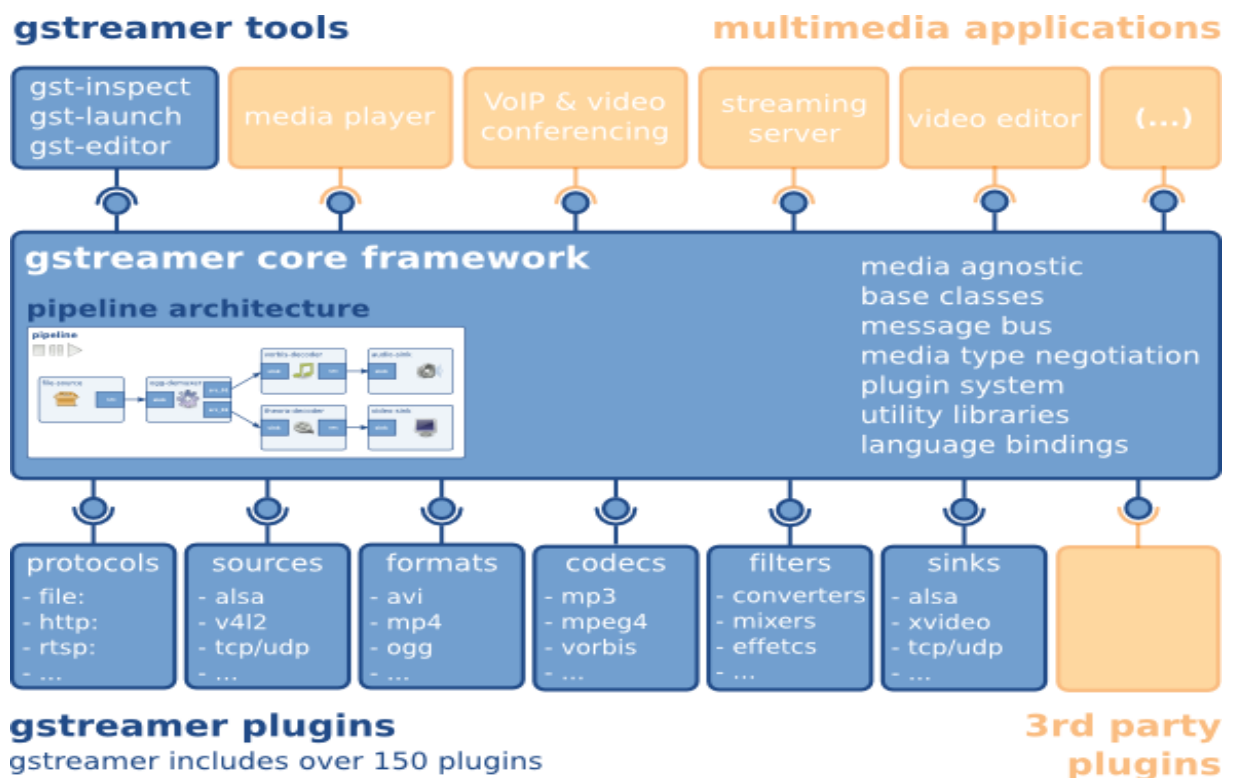


Figure 3.1. GStreamer overview

3.2. Android Stagefright

Stagefright is a new media framework that supports local playback and HTTP progressive streaming this introduced in Android 2.2. It seems to be simple and straightforward compared with the OpenCORE solution which was used in previous Android releases.

Stagefright is a dedicated framework in Android platform for handling media local playback and streaming tasks. So Stagefright use less models and simple ways to process media data. (An overview of Stagefright player 2010).

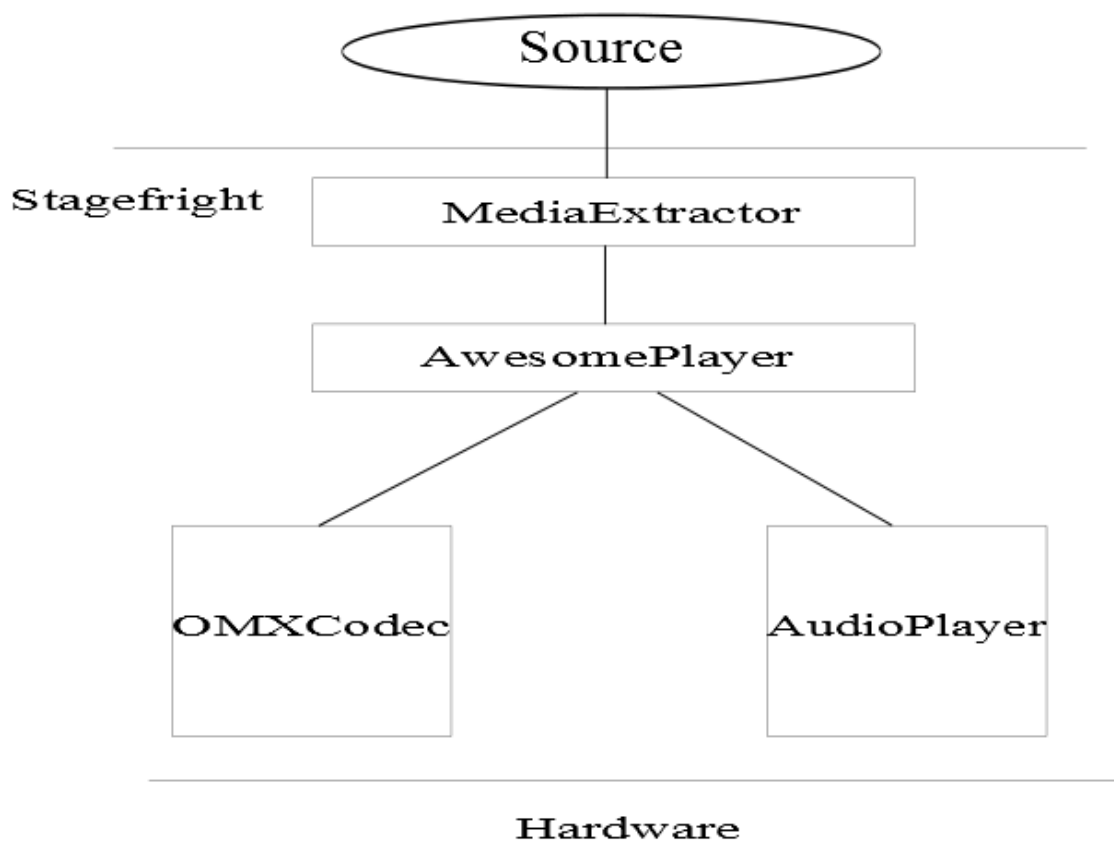


Figure 3.2. Stagefright overview

MediaExtractor is responsible for retrieving track data and the corresponding metadata from the underlying file system or http stream.

Leveraging OMX for decoding: there are two OMX plugins currently, adapting to PV's software codec and vendor's hardware implementation respectively. And there is a local implementation of software codecs which encapsulates PV's decoder APIs directly.

AudioPlayer is responsible for rendering audio. It also provides the time-based for timing and A/V synchronization whenever audio track is present.

Depending on which codec is picked, a local or remote render will be created for video rendering; and system clock is used as the time-based for video only playback.

AwesomePlayer works as the engine to coordinate the above modules, and is finally connected into android media framework through the adapter of Stagefright Player.

3.3. Microsoft Silverlight Media Framework

Microsoft's open source Silverlight Media Framework enables developers to quickly deploy a robust, scalable, customizable media player for IIS Smooth Streaming delivery. The SMF builds on the core functionality of the Smooth Streaming Client and adds a large number of additional features, including an extensibility API that allows developers to create plugins for the framework. The SMF also now includes full support for Windows Phone 7 so developers can incorporate high-end video playback experiences in their Windows Phone 7 applications. (Microsoft Media Platform: Player Framework).



Figure 3.3. Silverlight Media Framework overview

The latest release is SMFv2 which introduces a whole new, more modular architecture with selectively exclude libraries that aren't needed in their projects in order to avoid unnecessary file size increases. The SMFv2 framework will include plug-ins for Timed Text, URL frame linking, a metadata framework, support for the Microsoft Silverlight Analytics Framework, support for multiple audio tracks, improved bitrate monitoring, support for Silverlight 4 global styling, a JavaScript API, a logging plug-in, and a host of other new features. It also takes advantage of the new Managed Extensibility Framework (MEF) in Silverlight 4.

4. COMPARSION OF MULTIMEDIA FRAMEWORKS

4.1. Design Architecture

4.1.1. Open Source GStreamer

GStreamer fundamental design comes from the video pipeline at Oregon Graduate Institute as well as some ideas from DirectShow. GStreamer processes media by connecting a number of processing elements into a pipeline. Each element is provided by a plugin. Elements can be grouped into bins, which can be further aggregated thus forming a hierarchical graph. Elements communicate by means of pads. A source pad on one element can be connected to a sink pad on another. When the pipeline is in the playing state, data buffers flow from the source pad to the sink pad. Pads negotiate the kind of data that will be sent using capabilities with their connected peer pad.

Elements: Black box that can be chained to several other black boxes in order to create a data flow. For example, an element may be a file source, a de-multiplexer, a decoder or a display among many others.

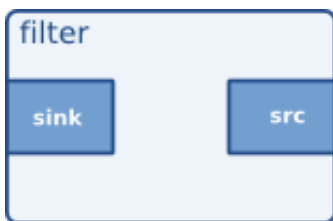


Figure 4.1. GStreamer element

Bins and Pipelines: Bin is a collection of elements and a pipeline is a type of bin that has the ability to allow execution of all its elements. A notable design aspect of bins is that they allow encapsulating logic for how to setup the contained graph, while the elements do the processing. So far in other frameworks, there is one place where one would specify the whole processing setup. In GStreamer one can write bins as reusable modules within the framework to split up the logic about setting up graph.

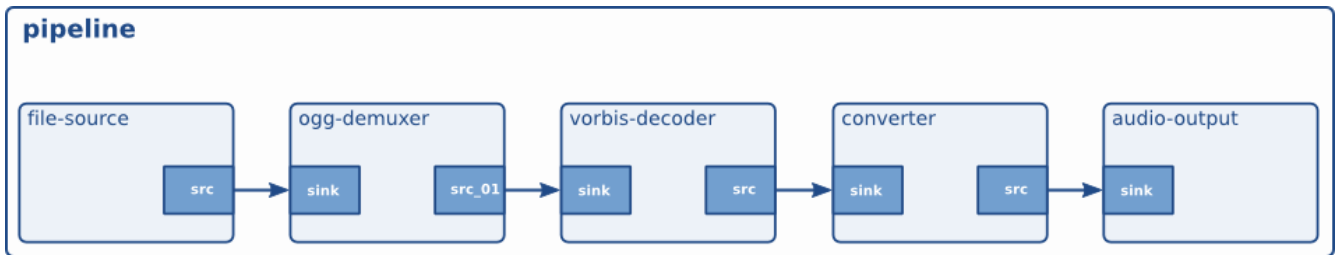


Figure 4.2. GStreamer pipeline

Pads: Allows communication between GStreamer elements. In Figure 4.3, a source element connects to a filter element by their pads. And the filter element does some processing and passes media data to sink element and they should be joined by pads. Pads can only be linked if the formats they both support have one or more in common.



Figure 4.3. GStreamer pad

Bus: The purpose of bus is to provide messages from the pipeline to the application. The pipeline is multi-threaded and the bus marshals the message into application main-loop. A bus need not be created since every pipeline has one bus. It only needs a message handler (a place to put the messages).

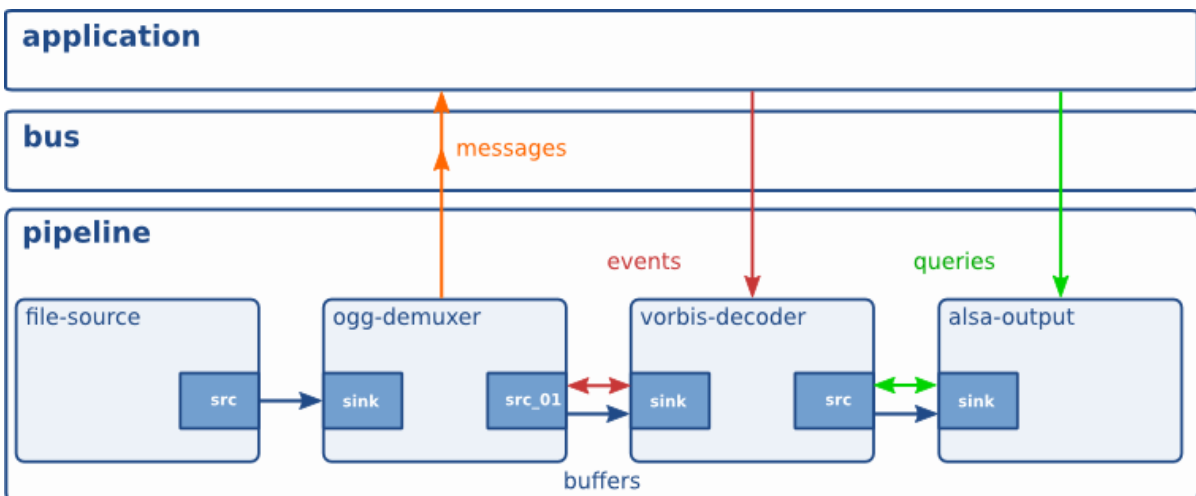


Figure 4.4. GStreamer bus

4.1.2. Android Stagefright

Multimedia architecture of the android is shown in Figure 4.5. Upper layer like media player can receive from different multimedia data. Through using the JNI Interface for *Jni_android_media_MediaPlayer*, multimedia data can connect with the lower layer component to perform the multimedia player service such as component *MediaPlayerService*. Finally, the service can create a new Stagefright player by using *creatPlayer()* function.

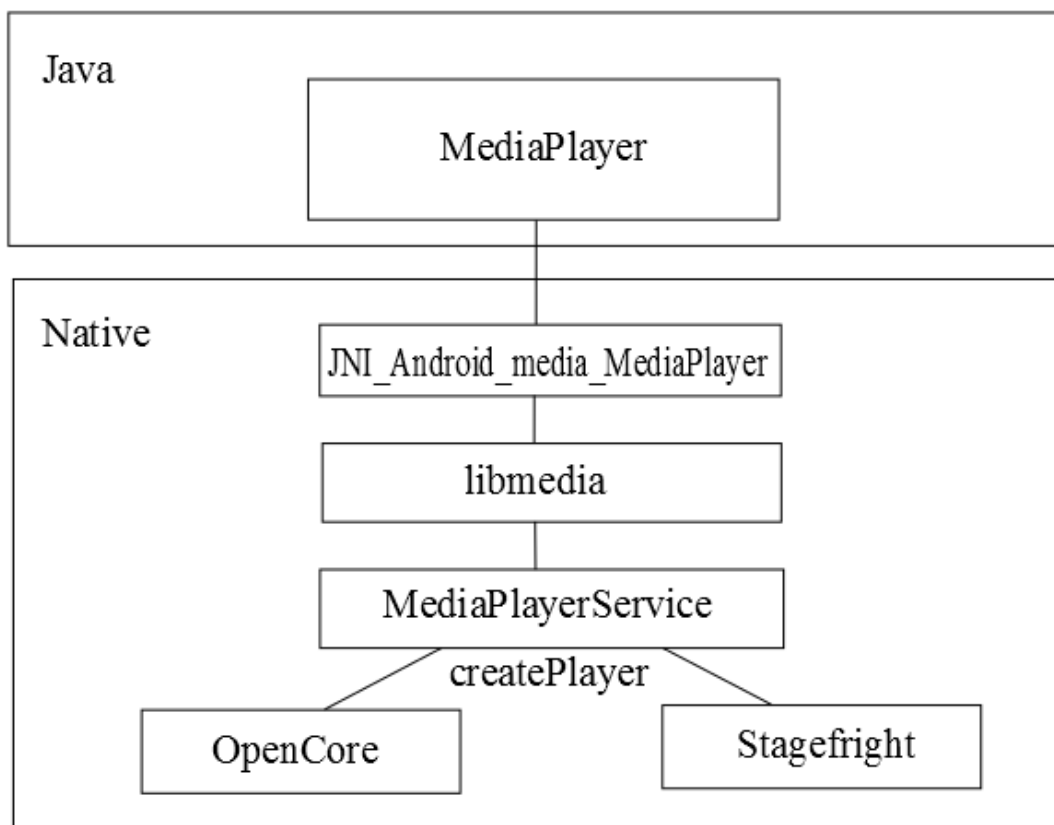


Figure 4.5. Android multimedia architecture

Media modules for android Stagefright consists of AwesomePlayer, MediaExtractor, OMXCodec, AudioPlayer and AwesomeRenderer as illustrated in Figure. AwesomePlayer provides many APIs so that application layer can use Java or JNI for multimedia playing. The use of the MediaExtractor module is mainly for video and audio separating. In the meanwhile, the OMXCodec module employs the OpenMAX framework, which is still an implementation of the OMX of OpenCORE. After the

decoding, video data is passed to AwesomeRenderer module for displaying, and the audio data is passed to AudioPlayer module for audio output. Currently, multimedia for android Stagefright can support different Codec formats. The Codec presents encode/decode of the video and audio. As shown in Table 1, decoder for video includes with H.263, H.264, MPEG-4 SP and VP8. Decoder for audio has the format supporting in AAC, MP3, etc. On the other hand, encoder supports for audio in H.263 and H.264, and encoder for audio is in formats AAC, AMR-NB and AMR-WB.

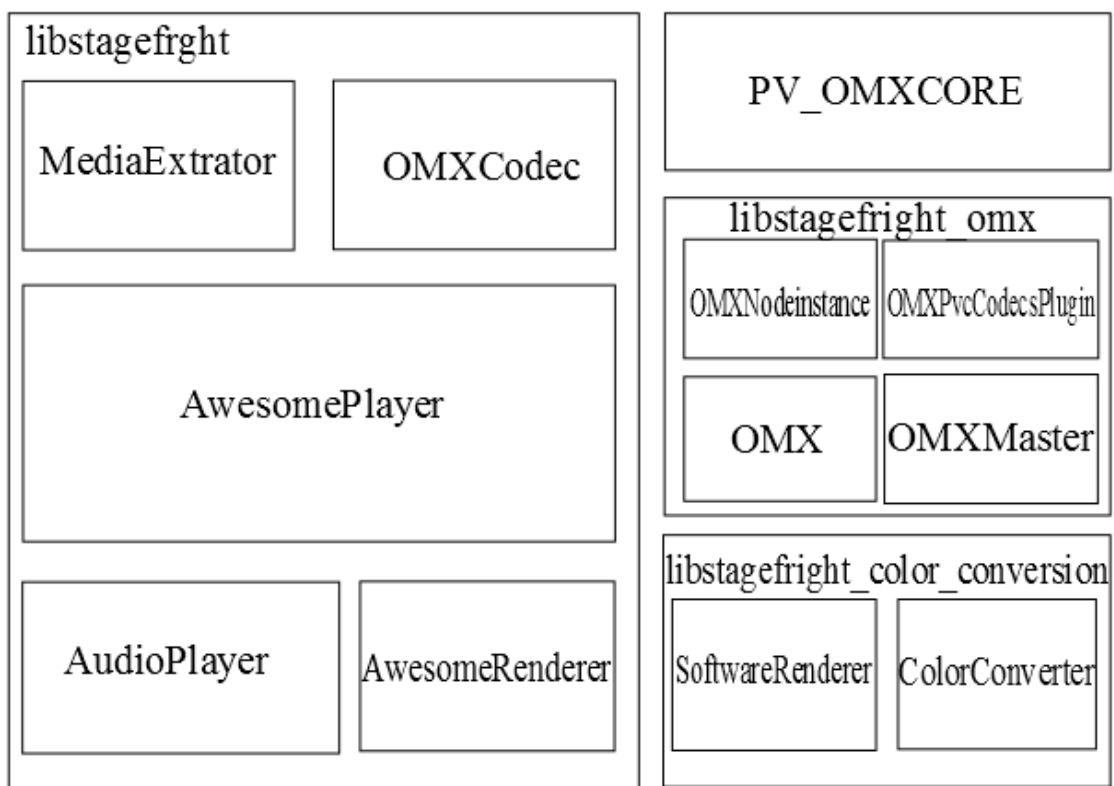


Figure 4.6. Stagefright internal models

Android Stagefright uses the shared library and AwesomePlayer module to manage both outputs for video and audio. AwesomePlayer also provides many APIs for application layer of android to connect with lower layer via calling JNI interface.

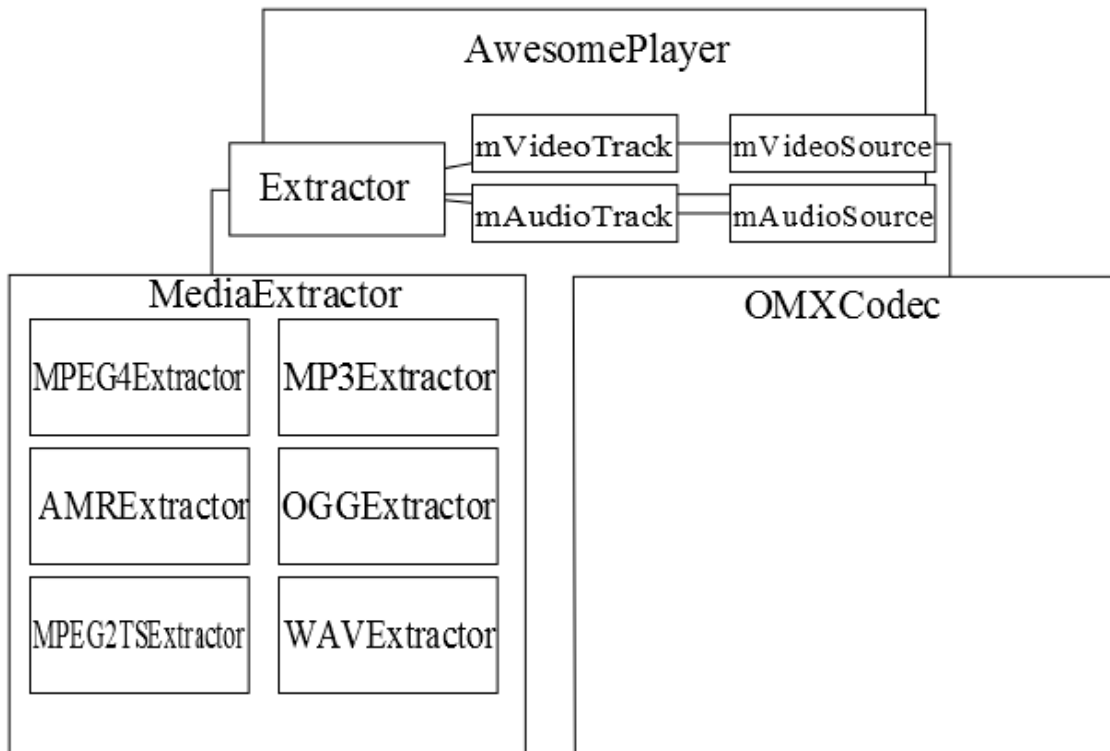


Figure 4.7. Stagefright architecture

The steps are presented in the following. A source file (media file) first parses multimedia format for audio and video to select its related media extractor component. The component for **MediaExtractor** has responsibility to divide data into two parts of video and audio, respectively. For the component in the track of media format can be found in *mVideoTrack*/*mAudioTrack*. Based on different media formats, *mVideoTrack* selects its video Codec format in *mVideoSource*, and *mAudioTrack* selects audio Codec format in *mAudioSource*. The component for **OMXCodec** must return the video decoding information to *mVideoSource*. The video decoding information is stored in component *kDecoderInfo*. Then component *mVideoTrack* catches information from *kDecoderInfo* to another component in *matchingCodecs*. Based on this sequence of *matchingCodecs*, hardware supported device must be first searched and software is for later. In this paper, we would trace Stagefright source code to find a decoding procedure of multimedia framework of android Stagefright which will be introduced as follows.

Firstly, if device checking belongs to a hardware decoding, then it will be assigned to a related OMX component as illustrated in Figure up. If the component is an OMX component, information will be returned, otherwise, it will keep searching as mentioned above as long as the hardware cannot be found. If the hardware searching has failed, a default searching will active, which means that the OMXCodec will adopt a software decoder, for example the AVC Decoder is a default solution. Finally, AwesomePlayer receives decoding data and transmits it to display module which is called *AwesomeRenderer*.

When an application requests for video and audio, besides of the video component is activated in OMXCodec, AudioPlayer is also created by AwesomePlayer on the same time. In the meanwhile, AudioPlayer is activated for reading audio data and it begins to trigger *AudioOutput*. In the same time, AudioPlayer sets function *callback()* to *AudioOutput*. Afterward, if function *callback()* is called, AudioPlayer would fetch decoding data from audio decoder. Finally, the component for *AudioFlinger* is responsible to provide the audio output

4.1.3. Microsoft Silverlight Media Framework

Silverlight Media Framework now a part of Microsoft Media Platform: Player Framework which is a feature-rich framework for developing media players in Silverlight. It contains a fairly comprehensive and skin able UI, and supports a reasonable amount of video formats and delivery methods out-of-the-box. (Microsoft Media Platform: Player Framework).



Figure 4.8. Silverlight media framework architecture

4.2. License and Legal

GStreamer core and most of plugins are released under GNU LGPL license that means developer and enterprise can use GStreamer core and plugins for commercial products and also can put proprietary plugins in sales box. The license strategy attracts more companies using GStreamer in commercial products which are a good driving force for GStreamer project and community. Nokia, Intel, Motorola, Texas instruments, Freescale, and many more have used GStreamer in their products.

Android Stagefright is a Google open source project which is under Apache license version 2.0. The license allows the user of the software the freedom to use the software for any purpose, to distribute, to modify and to redistribute modified version of the software but unlike most of other permission licenses it doesn't require modified versions of the software to be distributed using the same license.

The GPL/LGPL licensed projects force developer to release their contributions with same license. It imposes developers to give up any patents in the contributions. Apache license allows developers to use the code and make modification without contribution back. That's why Android industry companies are happy to adopt with Apache license. Because the companies can do whatever they want with the project's code and don't need to publish their modifications. However the drawback is that poor cooperation between users of the projects and incompatibilities rise for each fork of the projects. So from long term view, the projects will not work well. But it will give the companies a short term advantage by protecting their innovation and keeping their competitive positions.

Microsoft Silverlight Framework is an open source framework and source code is available for downloading in Microsoft codeplex web with Microsoft Public License (MS-PL). Be aware the license claims user can download the software source code but with no right to modify and distribute the modified software. That is the biggest difference with GStreamer and Stagefright from legal aspect. From here you also can find the difference between open source software and free software.

4.3. Developer Support

People get used to think Linux and open source software are for programming geeks and hackers. The use of such software can be quite difficult for normal users or developers who want to utilize it for particular purpose because it short of development documents, help manuals and technical supports. When they meet some problem they don't know where they can find answers for their particular questions. The adoption of GStreamer by companies like Nokia and Intel helped the project. They sponsored some amount of developer time, contributed code and tested the framework extensively. Now GStreamer is one of well-maintained open source projects. It offers different documents in GStreamer website. Application development document is for application developers who want to use GStreamer to create applications like media player, media format converter, media editor. Core design Documentation and Plugins references are for developers who are interested with inside GStreamer and want to contribute to it. Furthermore GStreamer mailing list and IRC channel are good place to ask particular questions people can meet GStreamer gurus who have rich of experience in multimedia, open source and embedded hardware.

Stagefright is a new and fresh designed media framework. Because the young age so far there is no public document available. But Google has pretty good high level API document in Android web for application developers and Stagefright source code can be downloaded from Google repository that can be used for developer and hardware vendors who want to study architecture and develop new codec and integrate to Stagefright. In internet there are plenty of forums and mailing list where people can ask.

Microsoft Silverlight Media Framework adds Windows Phone support after version 2.2. According Microsoft announced SMF is also a Microsoft's open source framework. Developers should be able to get SMF source code with Microsoft authorization. Microsoft always put API documents in MSDN so developer can get information to develop SMF based media application for Windows Phone platform. Microsoft offers rich of examples for illustration how to use SMF APIs and feature to accomplish your task in Windows Phone web as well. Developer can also find useful tutorial video clips from YouTube and Lean Visual Studio website. Since Microsoft is going to re-design

and rewrite whole middleware for Windows Phone 8 there is no much sense to pay too much attention to current SMF release but the change will not affect SMF high level APIs.

4.4. Implementation Language and Language binding

GStreamer is implemented with C language but with an object oriented fashion using GObject framework. It also has numerous language bindings like, C++, C#, Python, Java, Perl, Ruby and so on. Developers have rich of choices in programming languages and IDEs to write application on top of GStreamer. All the language support documents and examples are located in GStreamer web.

Android Stagefright is implemented with C++ language. User can find that in Stagefright repository from Google. The new features like codecs have to also align with the convention and it offers Java APIs for high level application which means UI application in Android like media streaming client, media player have to be implemented with Java. The popularity of Stagefright is undergoing there are no language binding support than Java and C++ so far. By the more and more widely adopt of Android platform I think the situation will change.

Microsoft Silverlight Media Framework is implemented with C++ language and offers high level APIs with C# and Visual Basic together with Silverlight IDE. Windows Phone UI applications are all implemented with C# or Visual Basic with Silverlight framework. Microsoft wants application developers to use a unique and centralize IDE to develop Windows Phone application that will dramatically decrease maintenance cost and bug fixing and feature deployment time.

4.5. Supported User Cases

In this section a simple media player implementations with the three MMFs' high level APIs are represented. Key sections of source codes will be showed and explained in the below and full building process and real device running demonstration can be found in appendix section.

4.5.1. GStreamer simple media player

First we need to initialize GStreamer.

```
gst_init(&argc, &argv);
```

Second we need to create a *playbin2* to handle media data processing task with following codes:

```
GstElement *pipeline = gst_element_factory_make("playbin2",
"player");
```

The *playbin2* is a top level GStreamer pipeline. It self-collects all necessary GStreamer elements and links them together for an audio and video player.

Third we need to set *uri* of media that we can specify the media file which will be played.

```
g_object_set(G_OBJECT(pipeline), "uri", uri, NULL);
```

Following code starts to play the media file:

```
gst_element_set_state(GST_ELEMENT(pipeline),
GST_STATE_PLAYING);
```

In this example a Glib *mainloop* is used for handling all available source of events.

```
GMainLoop *loop = g_main_loop_new(NULL, FALSE);
g_main_loop_run(loop);
```

So far we have all the components to create a simple media player in the code and we need to compile it and try. Following command will do for that purpose.

```
gcc `pkg-config --cflags --libs gstreamer-0.10` gst-
player.c -o gst-player
```

We can play now but still missing something. We need something that can communicate with pipeline and catch pipeline message, like if there is error happened or we already played the file till the end then the program can exit properly. For make that happen we are going to add following line to the code.

```

GstBus *bus =
gst_pipeline_get_bus(GST_PIPELINE(pipeline));
gst_bus_add_watch(bus, bus_call, NULL);
gst_object_unref(bus);
static gboolean
bus_call(GstBus *bus,
         GstMessage *msg,
         gpointer user_data)
{
    switch (GST_MESSAGE_TYPE(msg)) {
        case GST_MESSAGE_EOS: break;
        case GST_MESSAGE_ERROR: break;
        default: break;
    }

    return true;
}

```

A full steps example for creating GStreamer based media player which can be ran in Linux Desktop environment and Nokia N900 is illustrated in appendix.

4.5.2. SMF simple media player

There is a trend for UI application development that application's UI layout design is separate with UI function implementation. The benefit for that is UI designer can use XML liked file define the UI elements and layout without touch real implementation code that developers can focus on source code for function implementation. The changes in XML liked file will not affect implementation code and vice versa.

First we need define UI layout for our simple media player in Microsoft Silverlight SDK. The following code defines simple media player application orientation. The application supports portrait and landscape orientation. When user launches the application it will be first in portrait mode.

```
SupportedOrientations="PortraitOrLandscape"
Orientation="Portrait"
```

Second we need to use a media element to create simple player instance and reserve some area for it. The media element instance initiated with auto that means when the media clip is loaded by the application it starts to play automatically. We also create a progress bar instance in the player to show the play position.

```
<Grid x:Name="ContentPanel" Grid.Row="1"
Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="40" />
    </Grid.RowDefinitions>
    <MediaElement Name="myMediaElement" AutoPlay="True"
Grid.Row="0" />
    <ProgressBar Name="pbVideo" Grid.Row="1" />
</Grid>
```

Third we define some basic control elements like play, pause, and stop buttons in application bar area to control the playback.

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True"
IsMenuEnabled="True" >
```

```

        <shell:ApplicationBarIconButton
IconUri="/icons/play.png" Click="Play_Click"
Text="Play"/>

        <shell:ApplicationBarIconButton
IconUri="/icons/pause.png" Click="Pause_Click"
Text="Stop"/>

        <shell:ApplicationBarIconButton
IconUri="/icons/stop.png" Click="Stop_Click" Text="Stop"/>

    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>

```

After we create all necessary UI elements for simple media player the next step is to implement the UI functions. Like when user taps on pause button the playback should be paused and during the media playback user get a call when the call ends the playback should resume from stopped position.

```

void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    myMediaElement.MediaOpened += new
RoutedEventHandler(myMediaElement_MediaOpened);

    myMediaElement.MediaEnded += new
RoutedEventHandler(myMediaElement_MediaEnded);

    myMediaElement.CurrentStateChanged += new
RoutedEventHandler(myMediaElement_CurrentStateChanged);

    currentPosition.Tick += new
EventHandler(currentPosition_Tick);

    myMediaElement.Source = new
Uri("http://ecn.channel9.msdn.com/o9/ch9/4807/574807/ISWPE
05SLToolkitForWP_ch9.wmv", UriKind.Absolute);

```

```
    }  
  
    void myMediaElement_CurrentStateChanged(object sender,  
RoutedEventArgs e)  
    {  
        if (myMediaElement.CurrentState ==  
MediaElementState.Playing)  
        {  
            currentPosition.Start();  
  
            ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IsEnabled = false; // play  
  
            ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = true; // pause  
  
            ((ApplicationBarIconButton)ApplicationBar.Buttons[2]).IsEnabled = true; // stop  
        }  
  
        else if (myMediaElement.CurrentState ==  
MediaElementState.Paused)  
        {  
            currentPosition.Stop();  
  
            ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IsEnabled = true; // play  
  
            ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false; // pause  
  
            ((ApplicationBarIconButton)ApplicationBar.Buttons[2]).IsEnabled = true; // stop  
        }  
    }  
}
```

```

    }

    else

    {

        currentPosition.Stop();

        ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IsEnabled = true; // play

        ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false; // pause

        ((ApplicationBarIconButton)ApplicationBar.Buttons[2]).IsEnabled = false; // stop

    }

}

```

Now we have all the elements to create simple media player, UI layout and real behaviors. Then uses Microsoft Silverlight SDK compile it and run it with Windows Phone simulator before deploy to real device.

4.5.3. Stagefright simple media player

Like Windows Phone application Android use XML file to hold UI design and a Java file to keep real implementation codes. First we define our simple player UI in XML file. It looks like,

```

<?xml version="1.0" encoding="utf-8"?>

    <LinearLayout

        xmlns:android="http://schemas.android.com/apk/res/android"

        android:orientation="vertical"

        android:layout_width="fill_parent"

```

```

        android:layout_height="fill_parent"
        android:padding="5dp">

<ImageView
    android:id="@+id/thumbnail"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:layout_gravity="center"
    android:adjustViewBounds="true"
    android:maxHeight="175dp"
    android:minHeight="175dp"
    android:scaleType="fitCenter"
/>

<TextView
    android:gravity="center_vertical|center_horizontal"
    android:layout_height="wrap_content"
    android:text="TextView" android:layout_width="fill_parent"
    android:id="@+id/songName"></TextView>

<LinearLayout
    android:layout_height="wrap_content"
    android:id="@+id/linearLayout1"
    android:layout_width="fill_parent">

    <ToggleButton android:textOn=""
    android:textOff="" android:layout_marginLeft="5px"
    android:id="@+id/playPauseButton"
    android:layout_width="30px"
    android:background="@drawable/ic_play_pause"

```

```

android:layout_marginTop="5px"
android:layout_height="30px"></ToggleButton>

        <TextView android:id="@+id/musicCurrentLoc"
android:layout_marginLeft="5px"
android:layout_marginTop="5px"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></TextView>

        <SeekBar
android:layout_height="wrap_content"
android:layout_marginLeft="5px" android:layout_weight="1"
android:id="@+id/musicSeekBar"
android:layout_width="fill_parent"></SeekBar>

        <TextView android:id="@+id/musicDuration"
android:layout_marginLeft="5px"
android:layout_marginTop="5px"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></TextView>

        <ImageButton android:layout_marginLeft="5px"
android:layout_marginTop="5px"
android:src="@drawable/ic_music_shuffle"
android:layout_height="35px" android:layout_width="35px"
android:id="@+id/shufflebutton"></ImageButton>

    </LinearLayout>

</LinearLayout>

```

Then a media player instance will be created and media clip position should be specified. Control function to the media player instance like play, pause, and stop should be placed in same Java file as well and the codes should look like,

```
String url = "http://....."; // media clip URL here
```



```

MediaPlayer mediaPlayer = new MediaPlayer(); // media
player instance
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url); //feed the clip to player
mediaPlayer.prepare(); // wait the clip to be loaded
mediaPlayer.start(); // start to play it

```

After the coding finished like what we did for Maemo5 and Windows Phone devices we compile the codes in Android SDK and transfer it to real device for testing and playing.

4.6 Performance

One often asked question is regarding performance index for each multimedia framework. The question is hard to answer for mobile platforms because current there is not a universal mobile hardware platform that can support the three mobile platforms. And the three MMFs are designed with different considerations and structures. But I recorded the CPU consumption, memory consumption and start up time for the three simple media players in N900, Lumia 800 and Nexus One.

Table 5.1. Performance indexes for the three simple media player

	CPU Usage(%CPU)	Memory Usage(MB)	SMP Application Launch Time(seconds)
N900	26.1	9.7	3.38
Lumia 800	17.1	6.1	2.3
Nexus One	18.5	7.5	2.2

In Nokia N900 and Google Nexus One, the Linux commands *top* and *time* were used for getting the performance values. In Lumia 800, The *Windows Phone Marketplace Test Kit* and *SystemInfo* from Microsoft were used for collecting related values.

5. CONCLUSION

In this master's thesis project three major mobile multimedia frameworks have been chosen for studying, analysis and comparison. And design architecture, license and legal, implementation and binding language, developer support, and real user case experiment are the check points of this master thesis projects.

As result from chapter 4 show have shown that from architecture point of view GStreamer use pipeline structure to organize its elements. The structure is intuitive and clear for developers. Media data is processed like streams which flow into sink pad and out from source pad of each element to go through whole pipeline. Android Stagefright can be considered as a mini version or simplified GStreamer. Because it is designed only for local playback and streaming purpose in Android platform. The architecture of Stagefright is simple. Whenever a media file is feed, *MediaExtractor* parse audio and video part from the media file then pass them to *AwesomePlayer*. *AwesomePlayer* choose proper audio, video decoder to process the inflow audio track and video track and rendering them to destination hard wares. Silverlight media framework is like Stagefright only in charge playback tasks in Windows Phone 7 platform and it use a player interface to control and synchronize with underline models.

From license and legal aspect, the three MMFs use different licenses for their source code. GStreamer and Stagefright use similar license strategy that allows third part to modify and redistribute the code and also can accept private plugins. Silverlight media framework is a Microsoft product by the nature it against free software. So it use MS-PL license and even the source code is open for outside but third part cannot modify and redistribute the code.

The three MMFs all have good technical documents support for UI application developers. GStreamer has also good maintained plugin development document available in GStreamer main page. Stagefright and Silverlight media framework don't have the documented instruction for plugin development. If developer wants to implement new feature and integrate to the two MMF. The primary way is gone through the source codes and figure out inside modules relationships.

The three MMFs are implemented with popular programming language. Stagefright and SMF are implemented with C++ and GStreamer use C together with GObject. Because GStreamer is a cross platform MMF with long history it has more wide range language binding than the other two MMFs.

From the chapter 4 supported user case section, we can see that three MMFs all offer simple and intuitive APIs for handling media playback functions. UI application developer can use the APIs to development media player application with basic controls like play, pause, stop, seeking.

Major difficulty was to setup development environment for experiments the simple media player application for the three MMFs. Each environment needs to use different operating system, configuration, development libraries, tools, and testing hardware. Sometime it takes long time to fine tuning the code for fixing a small issue. Otherwise the code just doesn't work even you think everything is fine there. Stagefright and SMF are young to the market there are not many documents for them in the public resource. It also takes me long time to read their source code and collect resources to understand their architectures.

Finally the conclusion is GStreamer is an obvious powerful and mature open source multimedia framework. Not only the software has been developed and polished for more than 10 years by experience developers and testers but also it has already integrated with high quality plugins support wide rang demands and offers great extension abilities as well. Unlike other open source projects which are short of user documents and technical support, GStreamer has well maintained documents, user guides with examples for different level user. It has rich of plugins which are able to cover most of the use cases and the open source natural allows developers to study GStreamer architecture and easily develop a new plugin for their purpose like new audio, video codecs and multimedia container. Most importantly it has been driven and adopted by world famous IT companies used in their products. GStreamer license strategy make it is not only suitable for researching and prototyping projects but also flexible enough for deploying in commercial products.

Android Stagefright is a young age multimedia framework which start to use in Android 2.2 afterwards devices. Stagefright only can handle media playback related tasks if user wants to work on media post processing tasks like edit captured video he/she has to use other Android frameworks. Current there are not many Stagefright documents available in the Internet but only high level APIs. User only can study its architecture from Stagefright source code and develop a new component for Stagefright. It is also quite hard to port to any other platform than Android. If user wants to develop an Android multimedia project Stagefright is the number one option. GStreamer community is currently working on Android support. It offers user another choice to utilize multimedia resources in Android platform.

Microsoft Silverlight media framework added Windows Phone 7 support in 2011. It offers almost same high level APIs like Windows desktop application. That makes desktop developers can fast and easily develop Windows Phone 7 multimedia applications. Like Stagefright the Silverlight media framework can only handle media playback tasks. User has to use other Silverlight components for the tasks other than media local playback and streaming. Microsoft decide to design a new media framework in Windows Phone 8 Apollo release and the new framework will keep high level API compatible and more features. If user wants to do a multimedia project with Windows Phone platform then there is no other choice than SMF.

Comparison for the three MMFs from software loading time, memory consumption, and CPU consumption directions should be more precisely done when there is a common hard ware platform available that can support Maemo, Android and Windows Phone 7 platforms.

REFERENCE

GStreamer Application Development Manual (0.10.36) [online] [cited 2012-4-13].

Available from Internet: <URL:

<http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html>

>.

Microsoft Media Platform: Player Framework [online] [cited 2012-4-13].

Available from Internet: <URL: <http://smf.codeplex.com>>.

An overview of Stagefright player (2010) [online] [cited 2012-4-13].

Available from Internet: <URL: <http://freepine.blogspot.com/2010/01/overview-of-stagefrighter-player.html>>.

A Digital Media Primer for Geeks [online] [cited 2012-4-13].

Available from Internet: <URL: <http://xiph.org/video/vid1.shtml>>.

Matroska the extensible, open source, open standard multimedia container [online]

[cited 2012-4-13]. Available from Internet: <URL: <http://matroska.org/>>.

Documentation/Maemo 5 Developer Guide [online] [cited 2012-4-13].

Available from Internet: <URL:

http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide>.

Android developers [online] [cited 2012-4-13].

Available from Internet: <URL: <http://developer.android.com>>.

Windows Phone Development [online] [cited 2012-4-13].

Available from Internet: <URL: [http://msdn.microsoft.com/en-us/library/ff402535\(v=VS.92\).aspx](http://msdn.microsoft.com/en-us/library/ff402535(v=VS.92).aspx)>.

Jim Rasmusson, Fredrik Dahlgren, Harald Gustafsson & Tord Nilsson (2004)

Multimedia in Mobile Phones – The ongoing revolution [online] [cited 2012-4-

13]. Available from Internet: <URL:

http://www.ericsson.com/ericsson/corpinfo/publications/review/2004_02/files/2004122.pdf >.

WebM an open web media project [online] [cited 2012-4-13].

Available from Internet: <URL: <http://www.webmproject.org>>

iOS Developer Library [online] [cited 2012-4-13]

Available from Internet: <URL:

<http://developer.apple.com/library/ios/#documentation/AudioVideo/Conceptual/MultimediaPG/Introduction/Introduction.html>>

APPENDIX I

For installing Maemo5 SDK you should have a Linux machine which prefers running with Debian or Ubuntu release. Download SDK package from <http://maemo.org/development/> and following the installation instructions in the same page.

If everything goes fine you should have Maemo Fremantle root strap and scratchbox which is a cross compiler install in your computer. Current you have the basic development environment to write GUI applications for Nokia N900. (Documentation/Maemo 5 Developer Guide).

The environment in computer should look like this,

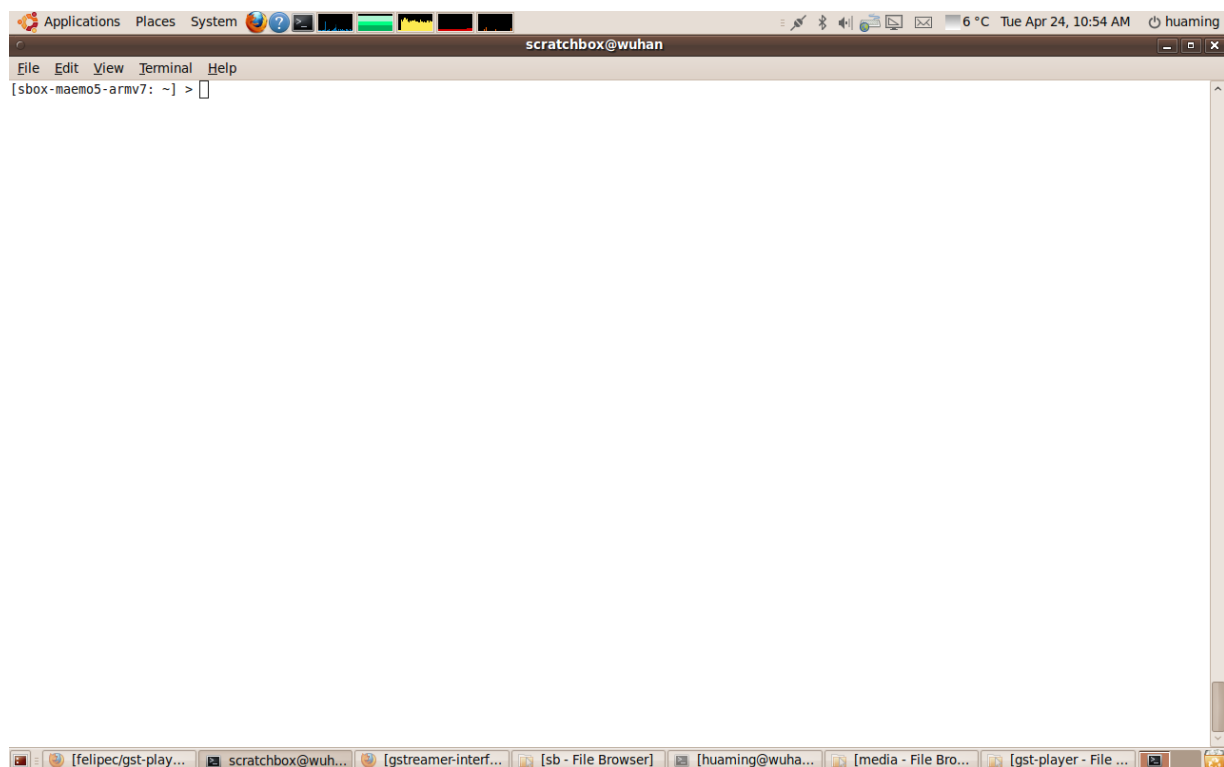


Figure I.1. Login maemo5 scratchbox

Make sure the compile target is for armv7 otherwise the compile code cannot be run in N900, to change the compile target run *sb-menu* in the environment.

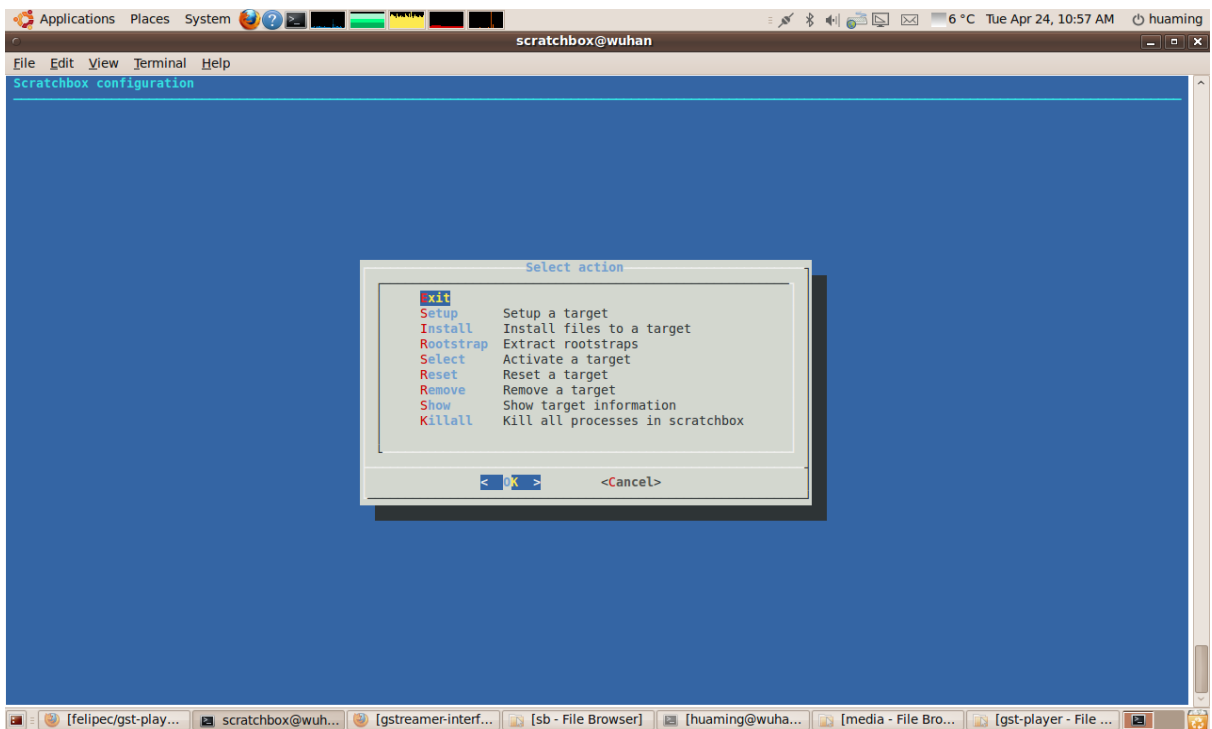


Figure I.2. Change compile target

Next step you should update the environment to catch up all the changes. Run *apt-get update* in your computer.

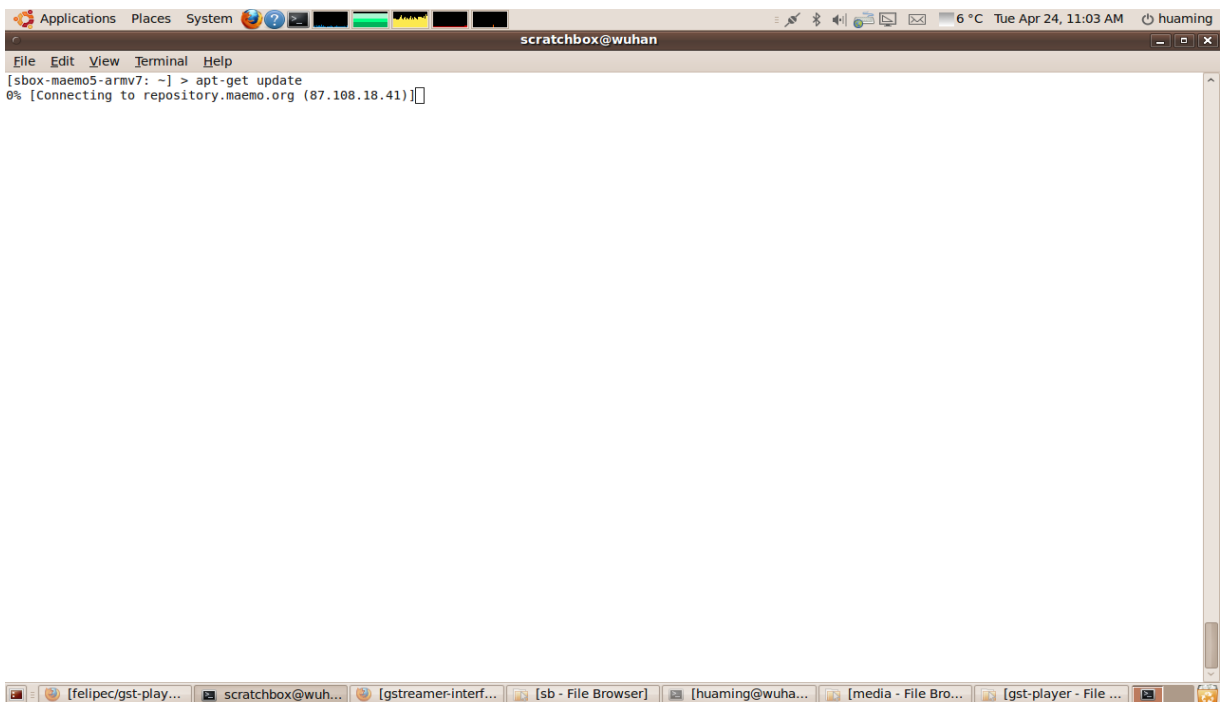
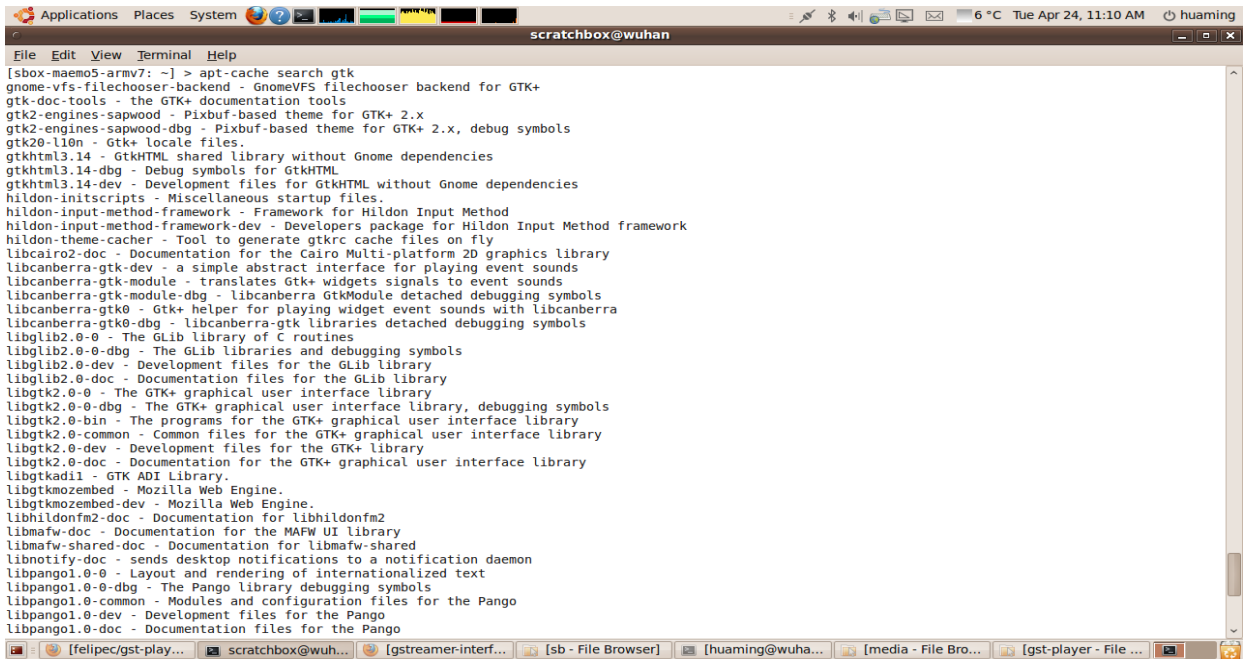


Figure I.3. Update scratchox development enviroment

The GStreamer simple player source codes separate into two part, UI codes and back end code. UI implementation is using GTK+ library and back end is using GStreamer libraries. Hence we need to install those libraries in computer as well. First run *apt-cache search gtk* to find GTK libraries packages name.



```

[sbox-maemo5-armv7: ~] > apt-cache search gtk
gnome-vfs-filechooser-backend - GnomeVFS filechooser backend for GTK+
gtk-doc-tools - the GTK+ documentation tools
gtk2-engines-sapwood - Pixbuf-based theme for GTK+ 2.x
gtk2-engines-sapwood-dbg - Pixbuf-based theme for GTK+ 2.x, debug symbols
gtk20-l10n - Gtk+ locale files.
gtkhtml3.14 - GtkHTML shared library without Gnome dependencies
gtkhtml3.14-dbg - Debug symbols for gtkhtml
gtkhtml3.14-dev - Development files for GtkHTML without Gnome dependencies
hildon-initscripts - Miscellaneous startup files.
hildon-input-method-framework - Framework for Hildon Input Method
hildon-input-method-framework-dev - Developers package for Hildon Input Method framework
hildon-theme-cacher - Tool to generate gtkrc cache files on fly
libcairo2-doc - Documentation for the Cairo Multi-platform 2D graphics library
libcanberra-gtk-dev - a simple abstract interface for playing event sounds
libcanberra-gtk-module - translates Gtk+ widgets signals to event sounds
libcanberra-gtk-module-dbg - libcanberra GtkModule detached debugging symbols
libcanberra-gtk0 - Gtk+ helper for playing widget event sounds with libcanberra
libcanberra-gtk0-dbg - libcanberra-gtk libraries detached debugging symbols
libglib2.0-0 - The GLib library of C routines
libglib2.0-0-dbg - The GLib libraries and debugging symbols
libglib2.0-dev - Development files for the glib library
libglib2.0-doc - Documentation files for the GLib library
libgtk2.0-0 - The GTK+ graphical user interface library
libgtk2.0-0-dbg - The GTK+ graphical user interface library, debugging symbols
libgtk2.0-bin - The programs for the GTK+ graphical user interface library
libgtk2.0-common - Common files for the GTK+ graphical user interface library
libgtk2.0-dev - Development files for the GTK+ library
libgtk2.0-doc - Documentation for the GTK+ graphical user interface library
libgtkadil - GTK ADI Library.
libgtkmozembed - Mozilla Web Engine.
libgtkmozembed-dev - Mozilla Web Engine.
libhildonfm2-doc - Documentation for libhildonfm2
libmaf-w - Documentation for the MAFW UI library
libmaf-shared-doc - Documentation for libmaf-shared
libnotify-doc - sends desktop notifications to a notification daemon
libpangol.0-0 - Layout and rendering of internationalized text
libpangol.0-0-dbg - The Pango library debugging symbols
libpangol.0-common - Modules and configuration files for the Pango
libpangol.0-dev - Development files for the Pango
libpangol.0-doc - Documentation files for the Pango

```

Figure I.4. Fetch GTK development libraries

And run *apt-cache search gstreamer* to find GStreamer development libraries packages name.

```

[sbox-maemo5-armv7: ~] > apt-cache search gstreamer
gstreamer-tools - Tools for use with GStreamer
gstreamer0.10-alsa - GStreamer plugin for ALSA
gstreamer0.10-alsa-dbg - Debug symbols for ALSA plugin
gstreamer0.10-doc - GStreamer core documentation and manuals
gstreamer0.10-gnomevfs - GStreamer plugin for GnomeVFS
gstreamer0.10-gnomevfs-dbg - Debug symbols for GnomeVFS plugin
gstreamer0.10-openmax - gst-openmax is a GStreamer plug-in that allows communication with
gstreamer0.10-plugins-bad - GStreamer plugins from the "bad" set
gstreamer0.10-plugins-bad-dbg - GStreamer plugins from the "bad" set
gstreamer0.10-plugins-bad-dev - Extra GStreamer plugins from the "bad" set
gstreamer0.10-plugins-bad-doc - GStreamer documentation for plugins from the "bad" set
gstreamer0.10-plugins-bad-extra - Extra GStreamer plugins from the "bad" set
gstreamer0.10-plugins-base - GStreamer plugins from the "base" set
gstreamer0.10-plugins-base-apps - GStreamer helper programs from the "base" set
gstreamer0.10-plugins-base-dbg - GStreamer plugins from the "base" set
gstreamer0.10-plugins-base-doc - GStreamer documentation for plugins from the "base" set
gstreamer0.10-plugins-camera - GStreamer plugins for camera application
gstreamer0.10-plugins-camera-dbg - Debug symbols for gstreamer0.10-plugins-camera
gstreamer0.10-plugins-camera-dev - Development files for gstreamer0.10-plugins-camera
gstreamer0.10-plugins-camera-doc - GStreamer documentation for the camera plugins and libs
gstreamer0.10-plugins-good - GStreamer plugins from the "good" set
gstreamer0.10-plugins-good-dbg - GStreamer plugins from the "good" set
gstreamer0.10-plugins-good-doc - GStreamer documentation for plugins from the "good" set
gstreamer0.10-plugins-good-extra - Collection of various GStreamer plugins
gstreamer0.10-plugins-good-extra-dbg - Debug symbols for gstreamer0.10-plugins-good-extra
gstreamer0.10-sdl - GStreamer plugin for SDL output
gstreamer0.10-tools - Tools for use with GStreamer
gstreamer0.10-trace1ib - GStreamer performance monitor
gstreamer0.10-x - GStreamer plugins for X11
gstreamer0.10-x-dbg - Debug symbols for X11 plugins
libgdigicam-dev - GDigicam development files
libgdigicam-doc - GDigicam library documentation
libgdigicam-gst-camerabin-dev - GDigicam GStreamer CameraBin development files
libgdigicam-gst-camerabin-doc - GDigicam GStreamer CameraBin library documentation
libgdigicam-gst-camerabin0 - GDigicam GStreamer CameraBin library
libgdigicam-gst-camerabin0-dbg - Debug GStreamer CameraBin symbols for GDigicam library
libgdigicam0 - GDigicam library
libgdigicam0-dbg - Debug symbols for GDigicam library
libgstreamer-plugins-base0.10-0 - GStreamer libraries from the "base" set
libgstreamer-plugins-base0.10-0-dbg - Debug symbols for ALSA plugin

```

Figure I.5. Fetch GStreamer development libraries

In our example the `libgtk2.0-dev` and `libgstreamer-plugins-base0.10-dev` are what we need. So run `apt-get install libgtk2.0-dev libgstreamer-plugins-base0.10-dev` in your computer.

```

[sbox-maemo5-armv7: ~] > apt-get install libgtk2.0-dev libgstreamer-plugins-base0.10-dev
Reading package lists... Done
Building dependency tree... Done
libgtk2.0-dev is already the newest version.
libgstreamer-plugins-base0.10-dev is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
[sbox-maemo5-armv7: ~] >

```

Figure I.6. Install GTK development libraries in scratchbox

When users compile codes in Maemo5 SDK they may face compiling errors. If the errors are because missing libraries then user can use same way to search and install missing libraries.

GStreamer simple player back end code,

```
#include <gst/gst.h>

#include <gst/interfaces/xoverlay.h>

#include "gst-backend.h"

static GstElement *pipeline;

static GstElement *bin;

static GstElement *videosink;

static gpointer window;

static GstSeekFlags seek_flags = GST_SEEK_FLAG_FLUSH |
GST_SEEK_FLAG_KEY_UNIT;

static gboolean bus_cb (GstBus *bus,
                        GstMessage *msg,
                        gpointer data)
{
    switch (GST_MESSAGE_TYPE (msg))
    {
        case GST_MESSAGE_EOS:
            {
                g_debug ("end-of-stream");
                break;
            }
    }
}
```

```
case GST_MESSAGE_ERROR:
    {
        gchar *debug;
        GError *err;

        gst_message_parse_error (msg, &err,
&debug);

        g_free (debug);

        g_warning ("Error: %s", err->message);
        g_error_free (err);
        break;
    }
default:
    break;
}

return TRUE;
}

void backend_init (int *argc,
                  char **argv[])
{
    gst_init (argc, argv);
}
```

```
void backend_set_window (gpointer window_)
{
    window = window_;
}

void backend_play (const gchar *filename)
{
    backend_stop ();

    pipeline = gst_pipeline_new ("gst-player");

    bin = gst_element_factory_make ("playbin", "bin");
    videosink = gst_element_factory_make ("xvimagesink",
"videosink");

    g_object_set (G_OBJECT (bin), "video-sink", videosink,
"flags", 99, NULL);

    gst_bin_add (GST_BIN (pipeline), bin);
    {
        GstBus *bus;

        bus = gst_pipeline_get_bus (GST_PIPELINE
(pipeline));

        gst_bus_add_watch (bus, bus_cb, NULL);
    }
}
```

```
gst_object_unref (bus);
}

{
gchar *uri;

if (gst_uri_is_valid (filename))
{
    uri = g_strdup (filename);
}
else if (g_path_is_absolute (filename))
{
    uri = g_filename_to_uri (filename, NULL,
NULL);
}
else
{
    gchar *tmp;
    tmp = g_build_filename (g_get_current_dir (),
filename, NULL);
    uri = g_filename_to_uri (tmp, NULL, NULL);
    g_free (tmp);
}

g_debug ("%s", uri);
```

```
        g_object_set (G_OBJECT (bin), "uri", uri, NULL);
        g_free (uri);
    }

    g_object_set (G_OBJECT (videosink), "force-aspect-
ratio", TRUE, NULL);

    if (GST_IS_X_OVERLAY (videosink))
    {
        gst_x_overlay_set_xwindow_id (GST_X_OVERLAY
(videosink), GPOINTER_TO_INT (window));
        gst_x_overlay_handle_events (GST_X_OVERLAY
(videosink), FALSE);
    }

    gst_element_set_state (pipeline, GST_STATE_PLAYING);
}

void backend_stop (void)
{
    if (pipeline)
    {
        gst_element_set_state (pipeline, GST_STATE_NULL);
        gst_object_unref (GST_OBJECT (pipeline));
        pipeline = NULL;
    }
}
```

```
    }  
}  
void backend_pause (void)  
{  
    gst_element_set_state (pipeline, GST_STATE_PAUSED);  
}  
  
void backend_resume (void)  
{  
    gst_element_set_state (pipeline, GST_STATE_PLAYING);  
}  
  
void backend_reset (void)  
{  
    gst_element_seek (pipeline, 1.0,  
                     GST_FORMAT_TIME,  
                     seek_flags,  
                     GST_SEEK_TYPE_SET, 0,  
                     GST_SEEK_TYPE_NONE,  
GST_CLOCK_TIME_NONE);  
}  
void backend_seek (gint value)  
{  
    gst_element_seek (pipeline, 1.0,  
                     GST_FORMAT_TIME,
```



```

        seek_flags,
        GST_SEEK_TYPE_CUR, value *
GST_SECOND,
        GST_SEEK_TYPE_NONE,
GST_CLOCK_TIME_NONE);
}

void backend_seek_absolute (guint64 value)
{
    gst_element_seek (pipeline, 1.0,
        GST_FORMAT_TIME,
        seek_flags,
        GST_SEEK_TYPE_SET, value,
        GST_SEEK_TYPE_NONE,
GST_CLOCK_TIME_NONE);
}

guint64 backend_query_position (void)
{
    GstFormat format = GST_FORMAT_TIME;
    gint64 cur;
    gboolean result;

    result = gst_element_query_position (pipeline,
&format, &cur);

    if (!result || format != GST_FORMAT_TIME)

```

```

        return GST_CLOCK_TIME_NONE;

    return cur;
}

guint64 backend_query_duration (void)
{
    GstFormat format = GST_FORMAT_TIME;

    gint64 cur;

    gboolean result;

    result = gst_element_query_duration (pipeline,
&format, &cur);

    if (!result || format != GST_FORMAT_TIME)
        return GST_CLOCK_TIME_NONE;

    return cur;
}

void backend_deinit (void)
{
}

```

GStreamer simple player back end header file,

```

#ifndef GST_BACKEND_H
#define GST_BACKEND_H

void backend_init (int *argc, char **argv[]);

```

```
void backend_deinit (void);  
void backend_set_window (gpointer window);  
void backend_play (const gchar *filename);  
void backend_stop (void);  
void backend_seek (gint value);  
void backend_seek_absolute (guint64 value);  
void backend_reset (void);  
void backend_pause (void);  
void backend_resume (void);  
guint64 backend_query_position (void);  
guint64 backend_query_duration (void);
```

```
#endif /* GST_BACKEND_H */
```

GStreamer simple player UI code,

```
#include <gtk/gtk.h>  
#include <gdk/gdkx.h>  
#include <gdk/gdkkeysyms.h>  
  
#include <string.h>  
  
#include "gst-backend.h"  
  
static gchar *filename;  
static GtkWidget *video_output;
```

```
static GtkWidget *pause_button;
static GtkWidget *scale;
static guint64 duration;
static GtkWidget *window;
static GtkWidget *controls;

#define DURATION_IS_VALID(x) (x != 0 && x != (guint64) -1)

static void
toggle_paused (void)
{
    static gboolean paused = FALSE;
    if (paused)
    {
        backend_resume ();
        gtk_button_set_label (GTK_BUTTON (pause_button),
"Pause");
        paused = FALSE;
    }
    else
    {
        backend_pause ();
        gtk_button_set_label (GTK_BUTTON (pause_button),
"Resume");
        paused = TRUE;
    }
}
```

```
    }  
}  
  
static void  
toggle_fullscreen (void)  
{  
    if (gdk_window_get_state (GTK_WIDGET (window)->window)  
== GDK_WINDOW_STATE_FULLSCREEN)  
    {  
        gtk_window_unfullscreen (window);  
        gtk_widget_show (controls);  
    }  
    else  
    {  
        gtk_window_fullscreen (window);  
        gtk_widget_hide (controls);  
    }  
}  
  
static void  
pause_cb (GtkWidget *widget,  
          gpointer data)  
{  
    toggle_paused ();  
}
```

```
static void
reset_cb (GtkWidget *widget,
          gpointer data)
{
    backend_reset ();
}

static gboolean
delete_event (GtkWidget *widget,
              GdkEvent *event,
              gpointer data)
{
    return FALSE;
}

static void
destroy (GtkWidget *widget,
         gpointer data)
{
    gtk_main_quit ();
}

static gboolean
```

```
key_press (GtkWidget *widget,  
          GdkEventKey *event,  
          gpointer data)  
{  
    switch (event->keyval)  
    {  
        case GDK_P:  
        case GDK_p:  
        case GDK_space:  
            toggle_paused ();  
            break;  
        case GDK_F:  
        case GDK_f:  
            toggle_fullscreen ();  
            break;  
        case GDK_R:  
        case GDK_r:  
            backend_reset ();  
            break;  
        case GDK_Right:  
            backend_seek (10);  
            break;  
        case GDK_Left:  
            backend_seek (-10);
```

```
        break;
    case GDK_Q:
    case GDK_q:
        gtk_main_quit ();
        break;
    default:
        break;
}

return TRUE;
}

static void
seek_cb (GtkRange *range,
        GtkScrollType scroll,
        gdouble value,
        gpointer data)
{
    guint64 to_seek;

    if (!DURATION_IS_VALID (duration))
        duration = backend_query_duration ();

    if (!DURATION_IS_VALID (duration))
```



```
        return;

        to_seek = (value / 100) * duration;

#if 0
        g_print ("value: %f\n", value);
        g_print ("duration: %llu\n", duration);
        g_print ("seek: %llu\n", to_seek);
#endif

        backend_seek_absolute (to_seek);
}

static void
start (void)
{
    GtkWidget *button;
    GtkWidget *vbox;

    window = GTK_WINDOW (gtk_window_new
(GTK_WINDOW_TOPLEVEL));

    g_signal_connect (G_OBJECT (window), "delete_event",
                     G_CALLBACK (delete_event), NULL);
```

```
g_signal_connect (G_OBJECT (window), "destroy",
                  G_CALLBACK (destroy), NULL);

g_signal_connect (G_OBJECT (window), "key-press-
event",
                  G_CALLBACK (key_press), NULL);

gtk_container_set_border_width (GTK_CONTAINER
(window), 0);

vbox = gtk_vbox_new (FALSE, 0);

gtk_container_add (GTK_CONTAINER (window), vbox);

gtk_widget_show (vbox);

controls = gtk_hbox_new (FALSE, 0);

gtk_box_pack_end (GTK_BOX (vbox), controls, FALSE,
FALSE, 2);

gtk_widget_show (controls);

{
    GdkColor color;
```

```
    gdk_color_parse ("black", &color);
    video_output = gtk_drawing_area_new ();
    gtk_widget_modify_bg (GTK_WIDGET (video_output),
GTK_STATE_NORMAL, &color);

    gtk_box_pack_start (GTK_BOX (vbox), video_output,
TRUE, TRUE, 0);

    gtk_widget_set_size_request (video_output, 0x200,
0x100);

    gtk_widget_show (video_output);
}

{
    button = gtk_button_new_with_label ("Pause");

    g_signal_connect (G_OBJECT (button), "clicked",
        G_CALLBACK (pause_cb), NULL);

    gtk_box_pack_start (GTK_BOX (controls), button,
FALSE, FALSE, 2);

    gtk_widget_show (button);
```

```
    pause_button = button;
}

{
    button = gtk_button_new_with_label ("Reset");

    g_signal_connect (G_OBJECT (button), "clicked",
                      G_CALLBACK (reset_cb), NULL);

    gtk_box_pack_start (GTK_BOX (controls), button,
FALSE, FALSE, 2);

    gtk_widget_show (button);
}

{
    GtkWidget *adjustment;
    adjustment = gtk_adjustment_new (0, 0, 101, 1, 5,
1);

    scale = gtk_hscale_new (GTK_ADJUSTMENT
(adjustment));

    gtk_box_pack_end (GTK_BOX (controls), scale, TRUE,
TRUE, 2);
```

```
        g_signal_connect (G_OBJECT (scale), "change-  
value",  
                           G_CALLBACK (seek_cb), NULL);
```

```
        gtk_widget_show (scale);  
    }
```

```
    gtk_widget_show (GTK_WIDGET (window));  
}
```

```
static gboolean
```

```
init (gpointer data)
```

```
{
```

```
    backend_set_window (GINT_TO_POINTER  
(GDK_WINDOW_XWINDOW (video_output->window)));
```

```
    if (filename)
```

```
        backend_play (filename);
```

```
    return FALSE;
```

```
}
```

```
static gboolean
```

```
timeout (gpointer data)
```

```
{  
  
    guint64 pos;  
  
    pos = backend_query_position ();  
    if (!DURATION_IS_VALID (duration))  
        duration = backend_query_duration ();  
  
    if (!DURATION_IS_VALID (duration))  
        return TRUE;  
  
#if 0  
    g_debug ("duration=%f", duration / ((double) 60 * 1000  
* 1000 * 1000));  
    g_debug ("position=%llu", pos);  
#endif  
  
    /** @todo use events for seeking instead of checking  
for bad positions. */  
    if (pos != 0)  
    {  
        double value;  
        value = (pos * ((double) 100) / duration);  
        gtk_range_set_value (GTK_RANGE (scale), value);  
    }  
}
```

```
    return TRUE;
}

int
main (int argc, char *argv[])
{
    gtk_init (&argc, &argv);
    backend_init (&argc, &argv);

    start ();

    if (argc > 1)
    {
        filename = g_strdup (argv[1]);
    }

    toggle_fullscreen ();
    g_idle_add (init, NULL);
    g_timeout_add (1000, timeout, NULL);

    gtk_main ();

    g_free (filename);
}
```

```

backend_deinit ();

return 0;
}

```

GStreamer simple player Makefile,

```

CC := gcc

EXTRA_WARNINGS := -Wextra -ansi -std=c99 -Wno-unused-
parameter

GST_LIBS := $(shell pkg-config --libs gstreamer-0.10
gstreamer-interfaces-0.10)

GST_CFLAGS := $(shell pkg-config --cflags gstreamer-0.10
gstreamer-interfaces-0.10)

GTK_LIBS := $(shell pkg-config --libs gtk+-2.0)

GTK_CFLAGS := $(shell pkg-config --cflags gtk+-2.0)

CFLAGS := -ggdb -Wall $(EXTRA_WARNINGS)

gst-player: ui.o gst-backend.o

gst-player: CFLAGS := $(CFLAGS) $(GTK_CFLAGS)
$(GST_CFLAGS)

gst-player: LIBS := $(LIBS) $(GTK_LIBS) $(GST_LIBS)

binaries += gst-player

all: $(binaries)

$(binaries):
    $(CC) $(LDFLAGS) $(LIBS) -o $@ $^

%.o:: %.c

```



```
$(CC) $(CFLAGS) -o $@ -c $<
```

clean:

```
rm -rf $(binaries)
```

```
find . -name "*.o" | xargs rm -rf
```

Compile GStreamer simple player source code in Maemo SDK then connect N900 to computer with usb cable. Copy the object file to device home directory and make sure there is a media clip in the same directory.

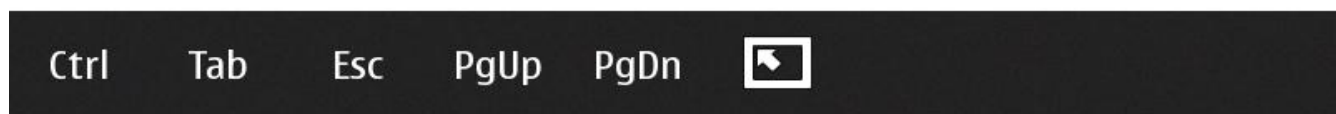
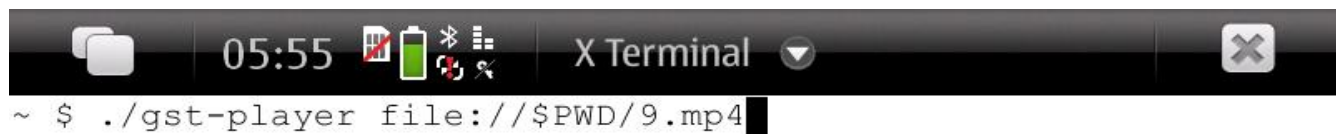


Figure I.7. Terminal command to run gst-player

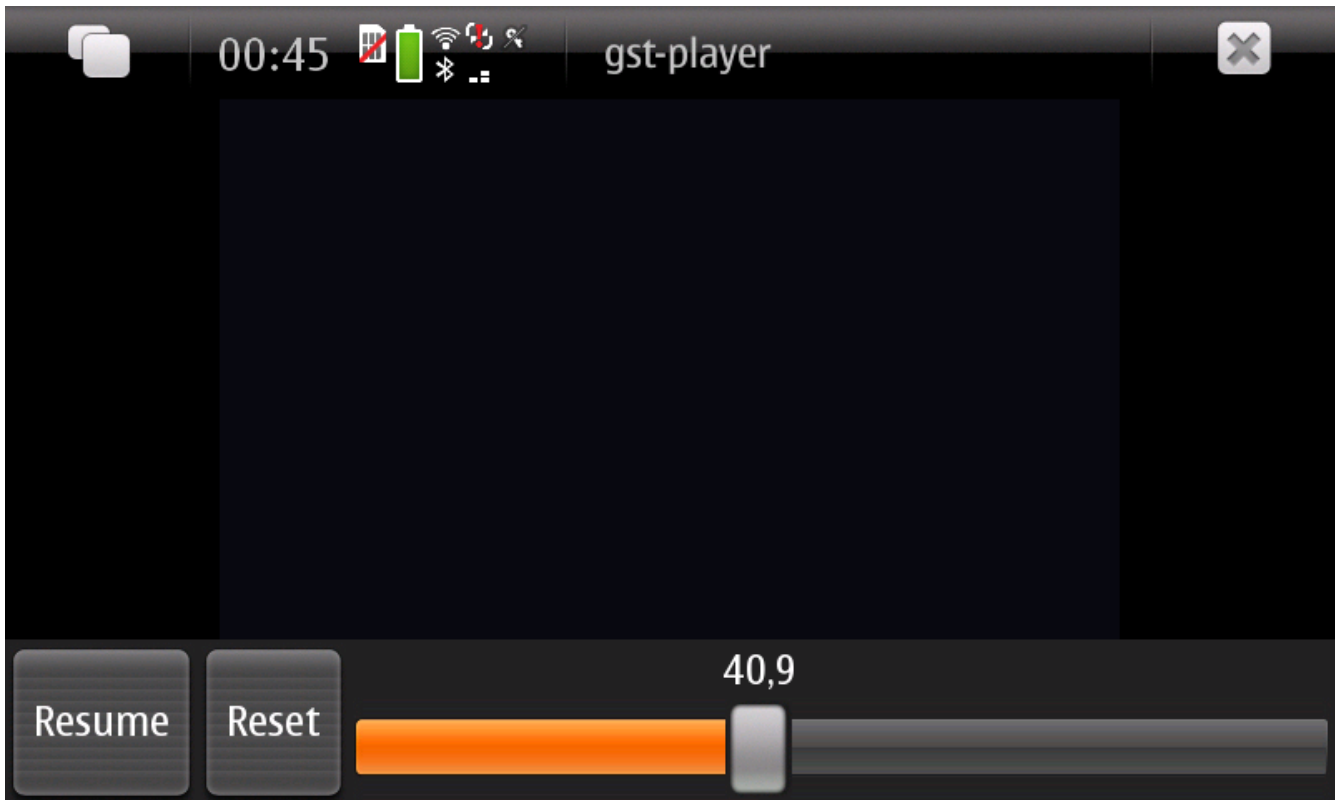


Figure I.8. GStreamer simple player UI layout

APPENDIX II

Download Windows Phone 7 SDK from <http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=27570> in a Windows computer.

Windows Phone SDK includes Visual Studio 2010 Express for Windows Phone, Windows Phone Emulator, XNA Game Studio, Expression Blend for Windows Phone, samples, and documentation. If Visual Studio 2010 Professional or later versions are already installed on your development computer, an add-in for Visual Studio 2010 Professional is automatically installed.

After Visual Studio 2010 is launched in computer the following window will display. Create a new project by selecting **the File | New Project** menu command. The New Project window will be displayed. Expand the Visual C# templates, and then select the Silverlight for Windows Phone templates. Select the Windows Phone Application template. Fill in the project Name as desired. (Windows Phone Development).

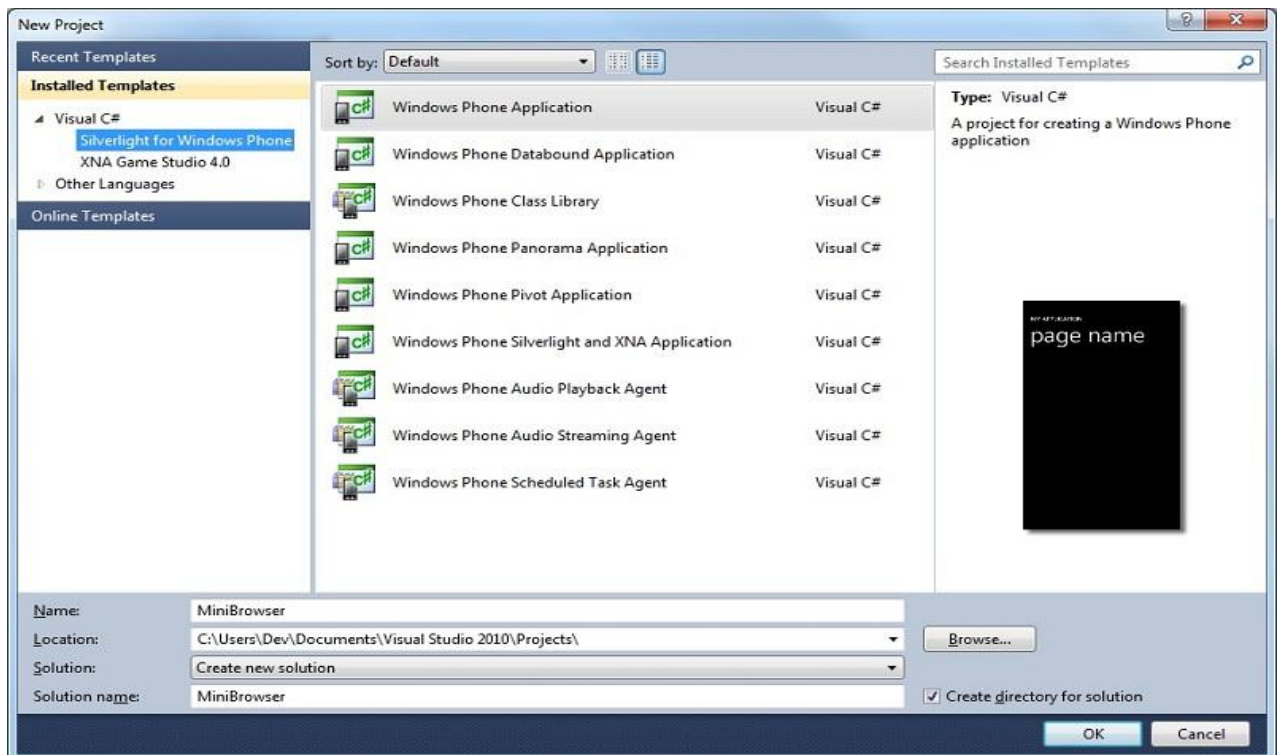


Figure II.1. Visual Studio 2010 main view

Click **OK**. The Windows Phone Platform selection window will appear. Select Windows Phone 7.1 for the Target Windows Phone Version.

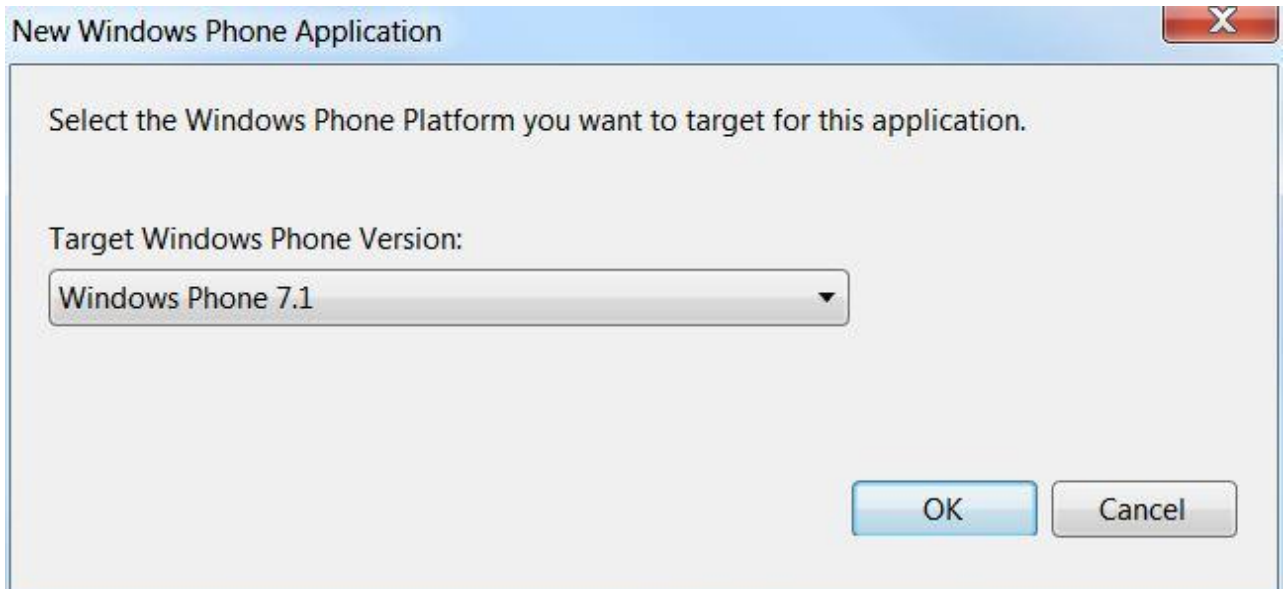


Figure II.2. Windows Phone 7 project

Copy the following code to the beginning part of *MainPage.xaml* file in simple media player project. The code defines simple media player can work on both landscape and portrait modes.

```
SupportedOrientations="PortraitOrLandscape"
Orientation="Portrait"
```

Also in *MainPage.xaml* add *MediaElement* control and Progress Bar control to *ContentGrid*. Set *AutoPlay* property of *MediaElement* to be true. Do not forget to add *RowDefinitions* to grid to make it possible to resize controls automatically in future.

```
<Grid x:Name="ContentPanel" Grid.Row="1"
Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="40" />
    </Grid.RowDefinitions>
```

```

        <MediaElement Name="myMediaElement" AutoPlay="True"
Grid.Row="0" />

        <ProgressBar Name="pbVideo" Grid.Row="1" />
</Grid>

```

Next step is to create an *ApplicationBar* in *MainPage.xaml*. *ApplicationBar* will contain 3 buttons to operate with media content: play, pause and stop. Copy the following codes to *ApplicationBar* section.

```

<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True"
IsMenuEnabled="True" >
        <shell:ApplicationBarIconButton
IconUri="/icons/play.png" Click="Play_Click"
Text="Play"/>
        <shell:ApplicationBarIconButton
IconUri="/icons/pause.png" Click="Pause_Click"
Text="Stop"/>
        <shell:ApplicationBarIconButton
IconUri="/icons/stop.png" Click="Stop_Click" Text="Stop"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>

```

Finally we add codes which really implement pre-defined functions and controls in *MainPage.xaml.cs* file.

```

public partial class MainPage : PhoneApplicationPage
{
    // variables and properties

```

```
DispatcherTimer currentPosition = new
DispatcherTimer();

// Constructor
public MainPage()
{
    InitializeComponent();
    this.Loaded += new
RoutedEventHandler(MainPage_Loaded);
}

void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    myMediaElement.MediaOpened += new
RoutedEventHandler(myMediaElement_MediaOpened);
    myMediaElement.MediaEnded += new
RoutedEventHandler(myMediaElement_MediaEnded);
    myMediaElement.CurrentStateChanged += new
RoutedEventHandler(myMediaElement_CurrentStateChanged);
    currentPosition.Tick += new
EventHandler(currentPosition_Tick);

    myMediaElement.Source = new
Uri("http://ecn.channel9.msdn.com/o9/ch9/4807/574807/ISWPE
05SLToolkitForWP_ch9.wmv", UriKind.Absolute);
}
```

```
void myMediaElement_CurrentStateChanged(object sender,
RoutedEventArgs e)
{
    if (myMediaElement.CurrentState ==
MediaElementState.Playing)
    {
        currentPosition.Start();

        ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IsEnabled = false; // play
        ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = true; // pause
        ((ApplicationBarIconButton)ApplicationBar.Buttons[2]).IsEnabled = true; // stop
    }

    else if (myMediaElement.CurrentState ==
MediaElementState.Paused)
    {
        currentPosition.Stop();

        ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IsEnabled = true; // play
        ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false; // pause
        ((ApplicationBarIconButton)ApplicationBar.Buttons[2]).IsEnabled = true; // stop
    }
}
```

```
else
{
    currentPosition.Stop();

    ((ApplicationBarIconButton)ApplicationBar.Buttons[0]).IsEnabled = true; // play

    ((ApplicationBarIconButton)ApplicationBar.Buttons[1]).IsEnabled = false; // pause

    ((ApplicationBarIconButton)ApplicationBar.Buttons[2]).IsEnabled = false; // stop
}
}
```

```
void myMediaElement_MediaEnded(object sender, RoutedEventArgs e)
{
    myMediaElement.Stop();
}
```

```
void myMediaElement_MediaOpened(object sender, RoutedEventArgs e)
{
    pbVideo.Maximum =
(int)myMediaElement.NaturalDuration.TimeSpan.TotalMilliseconds;

    myMediaElement.Play();
}
```

```
void currentPosition_Tick(object sender, EventArgs e)
```



```
{  
    pbVideo.Value = (int)myMediaElement.Position.TotalMilliseconds;  
}  
  
private void Play_Click(object sender, EventArgs e)  
{  
    myMediaElement.Play();  
}  
  
private void Pause_Click(object sender, EventArgs e)  
{  
    myMediaElement.Pause();  
}  
  
private void Stop_Click(object sender, EventArgs e)  
{  
    myMediaElement.Stop();  
}  
}
```

Compile and run the project in Visual Studio 2010 then the media player application launch in Windows Phone Emulator. You can see similar window like below.

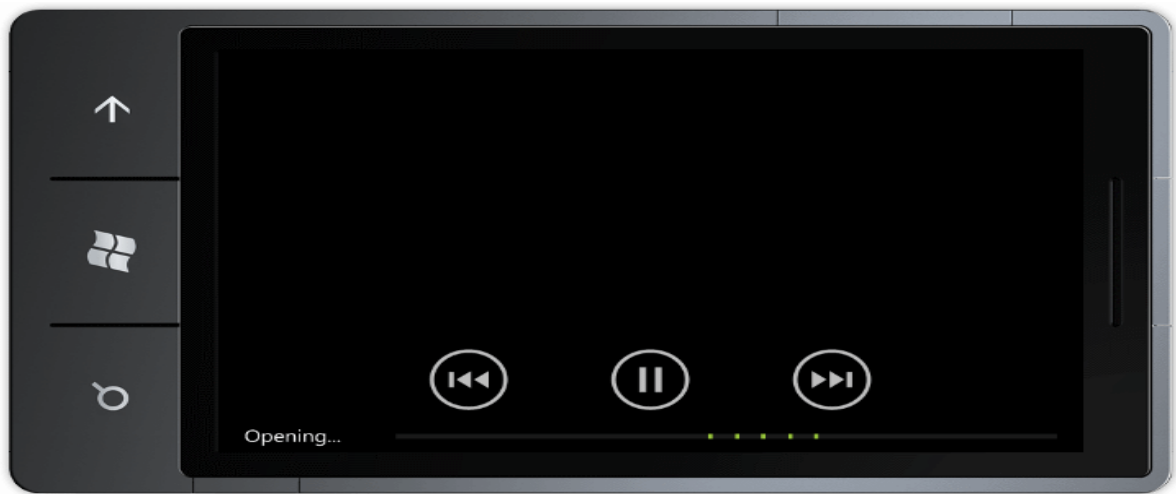


Figure II.3. Windows Phone 7 emulator

We are eager to try the simple player in real Windows Phone device now. In here a Nokia Lumia 800 is used as example but it should work in same way with all Windows Phone 7 based devices. One more thing before porting the application to Windows Phone terminal that user should make sure ZUNE is installed in same computer and target device is working in R&D mode to be able to install application for developing and debugging purposes. Connect the Windows Phone device via computer via a USB cable.

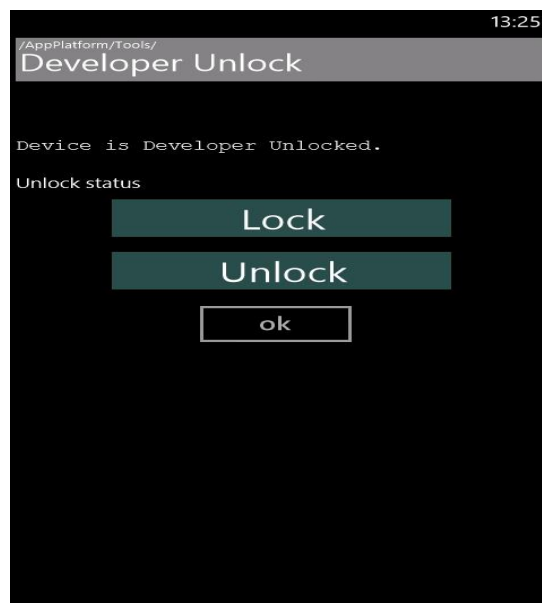


Figure II.4. Set Windows Phone device in Developer Unlock mode

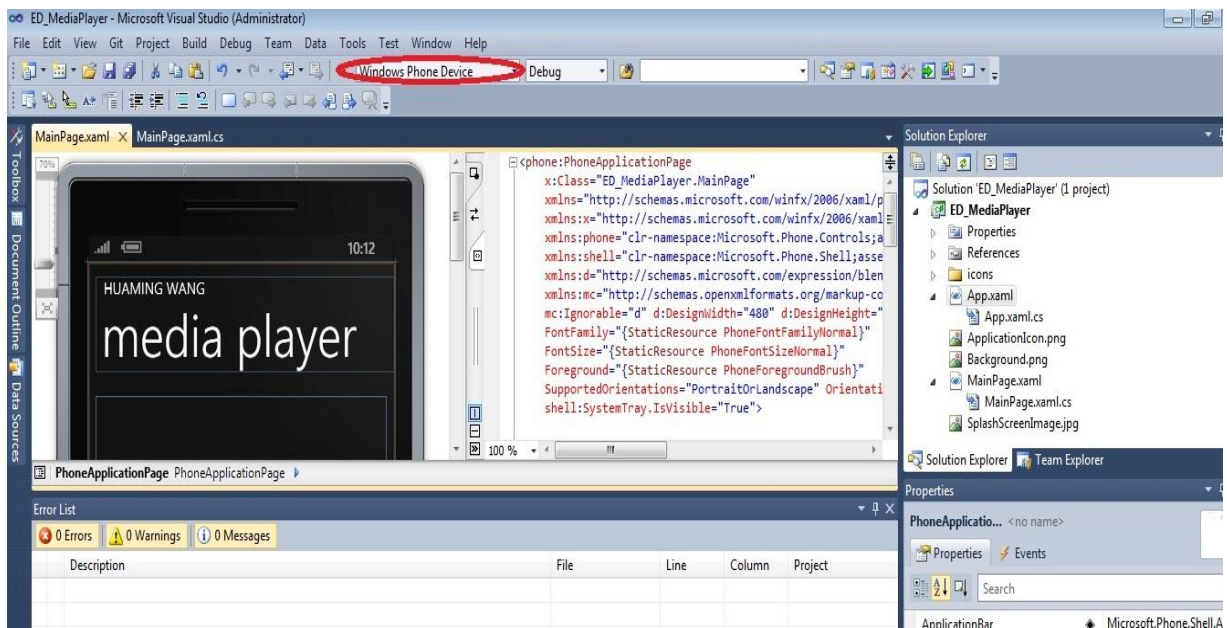


Figure II.5. Deploy project to Windows Phone device

Launch simple media player application in Lumia 800 and rotate the phone to see the application UI in portrait and landscape modes.



Figure II.6. Simple media player portrait mode in Lumia 800

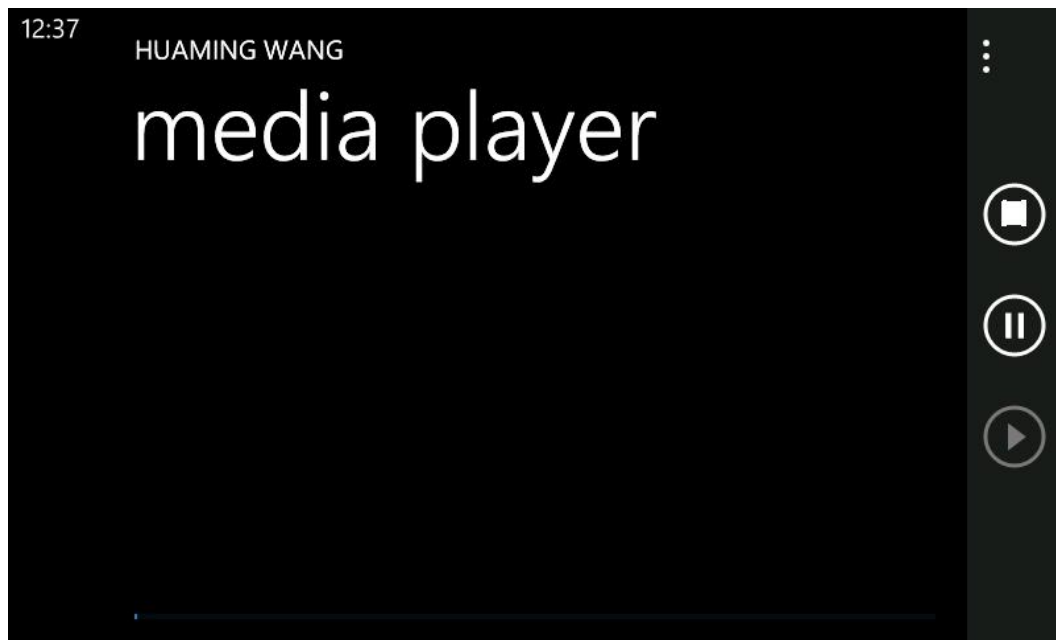


Figure II.7. Simple media player landscape mode in Lumia 800

APPENDIX III

Download Android SDK from <http://developer.android.com/sdk/index.html> and Android use Eclipse as IDE user needs to install it first in computer. Get "Eclipse Classic" version from <http://www.eclipse.org/downloads/>

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT) that is designed to give you a powerful, integrated environment in which to build Android applications. It extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, debug your applications using the Android SDK tools, and even export signed (or unsigned) APKs in order to distribute your application. In general, developing in Eclipse with ADT is a highly recommended approach and is the fastest way to get started with Android.

The SDK uses a modular structure that separates the major parts of the SDK—Android platform versions, add-ons, tools, samples, and documentation—into a set of separately installable packages. The SDK starter package, which you've already downloaded, includes only a single package: the latest version of the SDK Tools. To develop an Android application, you also need to download at least one Android platform and the associated platform tools. You can add other packages and platforms as well, which is highly recommended.

Using Windows installer, when you complete the installation wizard, it will launch the Android SDK Manager with a default set of platforms and other packages selected for you to install. Simply click **Install** to accept the recommended set of packages and install them.

Launch the Android SDK Manager from within Eclipse, select Window > Android SDK Manager. To download packages, use the graphical UI of the Android SDK Manager to browse the SDK repository and select new or updated packages. The Android SDK Manager installs the selected packages in your SDK environment. (Android developers).

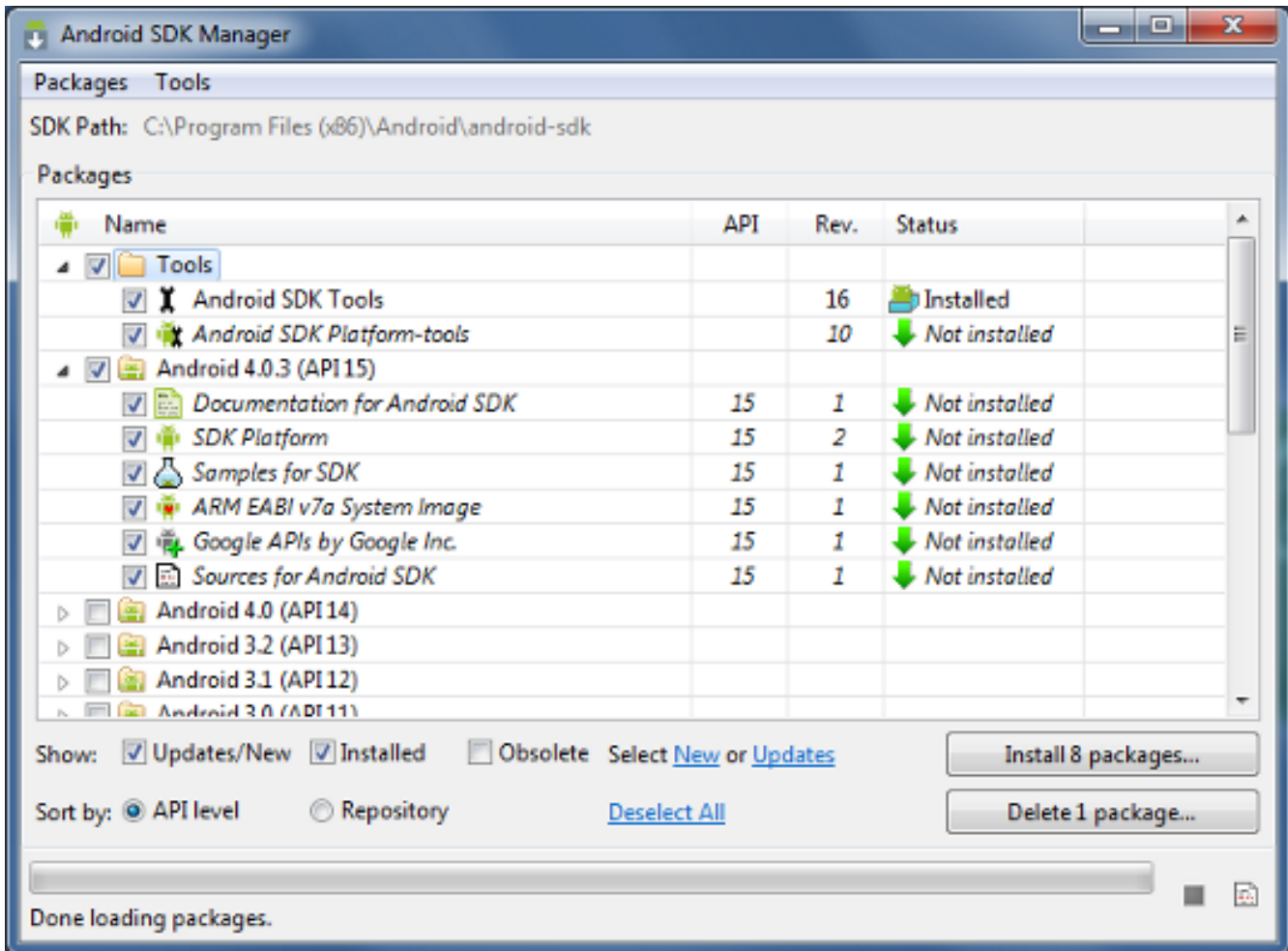


Figure III.1. Android SDK setup

It is simple to use the media player to play audio files. All we need to do is to initialize a media player object, set the audio stream, prepare the audio for playback and then finally start the playback. The following snippet demonstrates how to play an audio file obtained from an external URL. The media player takes care of streaming the audio automatically.

```
String url = "http://....."; // your URL here
```

```
MediaPlayer mediaPlayer = new MediaPlayer();
```

```
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
```

```
mediaPlayer.setDataSource(url);
```

```
mediaPlayer.prepare(); // might take long! (for buffering, etc)
```

```
mediaPlayer.start();
```

Prepare method prepares the audio for playback and can therefore take a long time. While the audio is preparing, the android's UI might seem to be non-functional and wouldn't respond to user requests. We will therefore use *prepareAsync()* method which prepares the media player in background on another thread and notifies the *onPreparedListener* when the prepare method is complete. Another thing that we want our media player to do is allow the user to work on other things while the media is playing in the background. This can be achieved using a background service rather than playing the music in the app's activity. Let's create a new Service by extending the Service class in android and associate build some functions to perform operations on media player.

```
Public class MusicService extends Service implements
MediaPlayer.OnPreparedListener, MediaPlayer.OnErrorListener,
MediaPlayer.OnBufferingUpdateListener {
```

```
    private static final String ACTION_PLAY = "PLAY";
```

```
    private static String mUrl;
```

```
    private static MusicService mInstance = null;
```

```
    private MediaPlayer mMediaPlayer = null; // The Media Player
```

```
    private int mBufferPosition;
```

```
    private static String mSongTitle;
```

```
    private static String mSongPicUrl;
```

```
    NotificationManager mNotificationManager;
```

```
    Notification mNotification = null;
```

```
    final int NOTIFICATION_ID = 1;
```



```

// indicates the state our service:

enum State {

    Retrieving, // the MediaRetriever is retrieving music

    Stopped, // media player is stopped and not prepared to play

    Preparing, // media player is preparing...

    Playing, // playback active (media player ready!). (but the media player
may actually be

                                // paused in this state if we don't have audio focus.
But we stay in this state

                                // so that we know we have to resume playback
once we get focus back)

    Paused

    // playback paused (media player ready!)

};

State mState = State.Retrieving;

@Override

public void onCreate() {

    mInstance = this;

    mNotificationManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);

}

```

```
@Override

public IBinder onBind(Intent arg0) {

    return null;

}

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

    if (intent.getAction().equals(ACTION_PLAY)) {

        mMediaPlayer = new MediaPlayer(); // initialize it here

        mMediaPlayer.setOnPreparedListener(this);

        mMediaPlayer.setOnErrorListener(this);

        mMediaPlayer.setOnBufferingUpdateListener(this);

mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);

        initMediaPlayer();

    }

    return START_STICKY;

}

private void initMediaPlayer() {

    try {

        mMediaPlayer.setDataSource(mUrl);

    } catch (IllegalArgumentException e) {
```

```

        // ...
    } catch (IllegalStateException e) {
        // ...
    } catch (IOException e) {
        // ...
    }

    try {
        mMediaPlayer.prepareAsync(); // prepare async to not block main
thread
    } catch (IllegalStateException e) {
        // ...
    }

    mState = State.Preparing;
}

public void restartMusic() {
    mState = State.Retrieving;
    mMediaPlayer.reset();
    initMediaPlayer();
}

protected void setBufferPosition(int progress) {
    mBufferPosition = progress;
}

```

```
}

/** Called when MediaPlayer is ready */

@Override

public void onPrepared(MediaPlayer player) {

    mState = State.Playing;

    mMediaPlayer.start();

    setUpAsForeground(mSongTitle + " (playing)");

}

@Override

public boolean onError(MediaPlayer mp, int what, int extra) {

    // TODO Auto-generated method stub

    return false;

}

@Override

public void onDestroy() {

    if (mMediaPlayer != null) {

        mMediaPlayer.release();

    }

    mState = State.Retrieving;

}
```

```
public MediaPlayer getMediaPlayer() {  
    return mMediaPlayer;  
}
```

```
public void pauseMusic() {  
    if (mState.equals(State.Playing)) {  
        mMediaPlayer.pause();  
        mState = State.Paused;  
        updateNotification(mSongTitle + " (paused)");  
    }  
}
```

```
public void startMusic() {  
    if (!mState.equals(State.Preparing) && !mState.equals(State.Retrieving))  
{  
        mMediaPlayer.start();  
        mState = State.Playing;  
        updateNotification(mSongTitle + " (playing)");  
    }  
}
```

```
public boolean isPlaying() {  
    if (mState.equals(State.Playing)) {  
        return true;  
    }  
}
```

```
    }  
    return false;  
}  
  
public int getMusicDuration() {  
    if (!mState.equals(State.Preparing) && !mState.equals(State.Retrieving))  
{  
        return mMediaPlayer.getDuration();  
    }  
    return 0;  
}  
  
public int getCurrentPosition() {  
    if (!mState.equals(State.Preparing) && !mState.equals(State.Retrieving))  
{  
        return mMediaPlayer.getCurrentPosition();  
    }  
    return 0;  
}  
  
public int getBufferPercentage() {  
    return mBufferPosition;  
}  
  
public void seekMusicTo(int pos) {
```

```
        if (mState.equals(State.Playing) || mState.equals(State.Paused)) {  
            mMediaPlayer.seekTo(pos);  
        }  
    }  
}
```

```
public static MusicService getInstance() {  
    return mInstance;  
}
```

```
public static void setSong(String url, String title, String songPicUrl) {  
    mUrl = url;  
    mSongTitle = title;  
    mSongPicUrl = songPicUrl;  
}
```

```
public String getSongTitle() {  
    return mSongTitle;  
}
```

```
public String getSongPicUrl() {  
    return mSongPicUrl;  
}
```

```
@Override
```

```

public void onBufferingUpdate(MediaPlayer mp, int percent) {
    setBufferPosition(percent * getMusicDuration() / 100);
}

/** Updates the notification. */
void updateNotification(String text) {
    PendingIntent pi =
        PendingIntent.getActivity(getApplicationContext(), 0,
new Intent(getApplicationContext(), MusicActivity.class),

    PendingIntent.FLAG_UPDATE_CURRENT);

    mNotification.setLatestEventInfo(getApplicationContext(),
getResources().getString(R.string.app_name), text, pi);

    mNotificationManager.notify(NOTIFICATION_ID, mNotification);
}

/**
 * Configures service as a foreground service. A foreground service is a service
that's doing something the user is
 * Actively aware of (such as playing music), and must appear to the user as a
notification. That's why we create
 * the notification here.
 */
void setUpAsForeground(String text) {
    PendingIntent pi =

```



```

        PendingIntent.getActivity(getApplicationContext(), 0,
new Intent(getApplicationContext(), MusicActivity.class),

        PendingIntent.FLAG_UPDATE_CURRENT);

        mNotification = new Notification();

        mNotification.tickerText = text;

        mNotification.icon = R.drawable.ic_mshuffle_icon;

        mNotification.flags |= Notification.FLAG_ONGOING_EVENT;

        mNotification.setLatestEventInfo(getApplicationContext(),
getResources().getString(R.string.app_name), text, pi);

        startForeground(NOTIFICATION_ID, mNotification);

    }
}

```

The *onCreate* method is called whenever a new service is created in android. We want our application to be able to communicate with the activity that builds the UI on android. Therefore, we store the instance of service as a static object in the service itself. Another alternative to this is to bind the service to the activity. But that's a bit complex to achieve and I would skip that for now. Since our service will run in background even when the user has not opened our application, we should therefore notify the user about what the service is doing. We will achieve that through notifications which are displayed in the top bar in android. We register the service as the *onPreparedListener* for music player so that we may be notified about the state of music player.

The most important point with music player is that it has a very complex state diagram and there are several restrictions on which functions can be called in which states. We therefore need to keep track of the state that music player is in. We will use our own enum for that purpose. The *onBufferingUpdateListener* is used to notify the

user about the buffer progress of the song. That almost wraps up the backend that we need to play music. Now let's get on to the front end, i.e. our activity that builds the UI.

Android does provide an easy way to have media player controls in android using the **MediaController** class. This creates floating controls and takes care of interacting with the media player. However, it doesn't offer any functions to modify the look and feel of the UI. We would therefore use our manually created music player controls for our music player. As with all other activities in android, we would first need to prepare our layout for the activity. Let's call the layout music.xml and add it to res/layout in our project.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
```

```
    android:padding="5dp">
```

```
<ImageView
```

```
    android:id="@+id/thumbnail"
```

```
    android:layout_height="wrap_content"
```

```
        android:layout_width="wrap_content"
```

```
        android:layout_gravity="center"
```

```
        android:adjustViewBounds="true"
```

```
        android:maxHeight="175dp"
```

```
        android:minHeight="175dp"
```

```
        android:scaleType="fitCenter"
```

```

        />
        <TextView                android:gravity="center_vertical|center_horizontal"
android:layout_height="wrap_content"                android:text="TextView"
android:layout_width="fill_parent" android:id="@+id/songName"></TextView>

        <LinearLayout                android:layout_height="wrap_content"
android:id="@+id/linearLayout1" android:layout_width="fill_parent">

                <ToggleButton                android:textOn=""                android:textOff=""
android:layout_marginLeft="5px"                android:id="@+id/playPauseButton"
android:layout_width="30px"                android:background="@drawable/ic_play_pause"
android:layout_marginTop="5px" android:layout_height="30px"></ToggleButton>

                <TextView                android:id="@+id/musicCurrentLoc"
android:layout_marginLeft="5px"                android:layout_marginTop="5px"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></TextView>

                <SeekBar                android:layout_height="wrap_content"
android:layout_marginLeft="5px"                android:layout_weight="1"
android:id="@+id/musicSeekBar" android:layout_width="fill_parent"></SeekBar>

                <TextView                android:id="@+id/musicDuration"
android:layout_marginLeft="5px"                android:layout_marginTop="5px"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></TextView>

                <ImageButton                android:layout_marginLeft="5px"
android:layout_marginTop="5px"                android:src="@drawable/ic_music_shuffle"
android:layout_height="35px"                android:layout_width="35px"
android:id="@+id/shufflebutton"></ImageButton>

        </LinearLayout>

</LinearLayout>

```

This layout will give us a basic UI for our android app. If you are using eclipse for development, you can have a look at the graphical layout to see how it might look like:

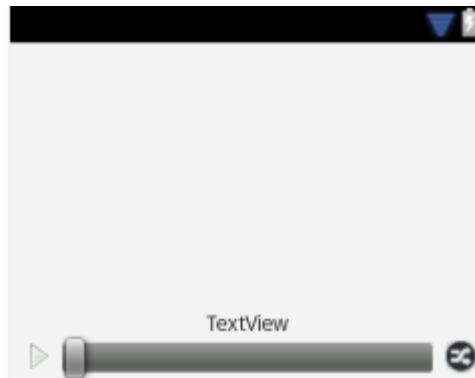


Figure III.2. Stagefright simple player UI layout

The toggle button will act as the button for play/pause. The xml selector will take care of changing the image of the button as per its state. Let's define the selector in `res/drawable/ic_play_pause.xml`

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true"
        android:drawable="@drawable/ic_music_pause" /> <!-- pressed -->
    <item android:drawable="@drawable/ic_music_play" /> <!-- default/unchecked -->
</selector>
```

We can now associate `onClick` action to the button to perform play/pause. Add the following code to your activity's `onCreate` method:

```
playPauseButton = (ToggleButton) findViewById(R.id.playPauseButton);
```

```
playPauseButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // Perform action on clicks
    }
});
```

```

        if (playPauseButton.isChecked()) { // Checked -> Pause icon visible
            start();
        } else { // Unchecked -> Play icon visible
            pause();
        }
    }
});

```

Now we need to associate our seek bar with the music player. For this, we need to implement the `OnSeekBarChangeListener` to call respective functions in the media player. Have your activity implement the interface and then add the following code to the activity's `onCreate` method:

```

musicSeekBar = (SeekBar) findViewById(R.id.musicSeekBar);
musicSeekBar.setOnSeekBarChangeListener(this);

```

Implementing the interface would require you to have the following method defined in your activity:

```

@Override
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    if (fromUser) {
        seekTo(progress);
    }
}

```

```

@Override
public void onStartTrackingTouch(SeekBar seekBar) {
    // TODO Auto-generated method stub
}

```

```
}

```

```
@Override

```

```
public void onStopTrackingTouch(SeekBar seekBar) {

```

```
    // TODO Auto-generated method stub

```

```
}

```

We would also need to check the status of music every second and update seek bar manually. To do this, we start a new thread and poll the service for music progress, buffer progress and music duration every second.

```
new Thread(new Runnable() {

```

```
    @Override

```

```
    public void run() {

```

```
        int currentPosition = 0;

```

```
        while (!musicThreadFinished) {

```

```
            try {

```

```
                Thread.sleep(1000);

```

```
                currentPosition = getCurrentPosition();

```

```
            } catch (InterruptedException e) {

```

```
                return;

```

```
            } catch (Exception e) {

```

```
                return;

```

```
            }

```

```
            final int total = getDuration();

```

```
            final String totalTime = getAsTime(total);

```

```
            final String curTime = getAsTime(currentPosition);

```

```

        musicSeekBar.setMax(total);        //song duration
        musicSeekBar.setProgress(currentPosition);    //for current
song progress

        musicSeekBar.setSecondaryProgress(getBufferPercentage());
// for buffer progress

        runOnUiThread(new Runnable() {

            @Override

            public void run() {

                if (isPlaying()) {

                    if (!playPauseButton.isChecked()) {

playPauseButton.setChecked(true);

                    }

                } else {

                    if (playPauseButton.isChecked()) {

playPauseButton.setChecked(false);

                    }

                }

                musicDuration.setText(totalTime);

                musicCurLoc.setText(curTime);

            }

        });

    }

}

}).start();

```

Notice the `runOnUiThread` method. This method is necessary to update activity's UI components in a thread-safe way. We have almost everything ready to have our first music player ready. But to play the music, we would need to start the music service.

```
MusicService.setSong(songUrl, songTitle, songPicUrl);  
startService(new Intent("PLAY"));
```



Figure III.3. Stagefright simple player in Nexus One