LIVE555 Streaming Media

This code forms a set of C++ libraries for multimedia streaming, using open standard protocols (RTP/RTCP, RTSP, SIP). These libraries - which can be compiled for Unix (including Linux and Mac OS X), QNX (and other POSIX-compliant systems) - can be used to build streaming applications. The libraries are already being used to implement applications such as the "LIVE555 Media Server", "LIVE555 Proxy Server", and "LIVE555 HLS Proxy" and "vobStreamer" (for streaming DVD content using RTP/RTCP/RTSP). The libraries can also be used to stream, receive, and process MPEG, H.265, H.264, H.263+, DV or JPEG video, and several audio codecs. They can easily be extended to support additional (audio and/or video) codecs, and can also be used to build basic RTSP or SIP clients and servers, and have been used to add streaming support to existing media player applications, such as "VLC" and "MPlayer". (For some specific examples of how these libraries can be used, see the **test programs** below.)

Source code

The project source code is available - as a ".tar.gz" file - here. See below for instructions on how to build it.
(Note: To use this software, you must be aware of how it is licensed, and your obligations under this license.)

Mailing list

There is a developers' mailing list: "live-devel@lists.live555.com". Users (or prospective users) of the libraries are encouraged to join this (low-volume) mailing list, and/or to review the mailing list's archives. (You can also search these archives using Google, by adding "site:lists.live555.com" to your search query.) **Before posting to the mailing list for the first time, please read the FAQ, to check if your question has already been answered.**

Support

The primary means of support for these libraries is the "live-devel@lists.live555.com" mailing list described above. (Note that you must first subscribe to the mailing list before you can post to it.)

Are you planning to implement RTP (and/or RTSP)? Instead of writing your own implementation from scratch, consider using these libraries. They have already been used in many real-world RTP-based applications, and are well-suited for use within embedded systems. The code includes an implementation of RTCP, and can easily be extended (via subclassing) to support new RTP payload types.

Help support improvements and extensions to the "LIVE555 Streaming Media" software: LIVE555 Funded Projects.

Description

The code includes the following libraries, each with its own subdirectory:

UsageEnvironment

The "UsageEnvironment" and "TaskScheduler" classes are used for scheduling deferred events, for assigning handlers for asynchronous read events, and for outputting error/warning messages. Also, the "HashTable" class defines the interface to a generic hash table, used by the rest of the code.

These are all abstract base classes; they must be subclassed for use in an implementation. These subclasses can exploit the particular properties of the environment in which the program will run - e.g., its GUI and/or scripting environment.

---

groupsock

The classes in this library encapsulate network interfaces and sockets. In particular, the "Groupsock" class encapsulates a socket for sending (and/or receiving) multicast datagrams.

---

liveMedia

This library defines a class hierarchy - rooted in the "Medium" class - for a variety of streaming media types and codecs.

BasicUsageEnvironment

This library defines one concrete implementation (i.e., subclasses) of the "UsageEnvironment" classes, for use in simple, console applications. Read events and delayed operations are handled using a select() loop.

---

testProgs

This directory implements some simple programs that use "BasicUsageEnvironment" to demonstrate how to develop applications using these libraries.

RTSP client

- **testRTSPClient** is a command-line program that shows you how to open and receive media streams that are specified by a RTSP URL - i.e., an URL that begins with *rtsp://*

  In this demonstration application, nothing is done with the received audio/video data. You could, however, use and adapt this code in your own application to (for example) decode and play the received data.

- **openRTSP** is similar to "testRTSPClient", but has many more features. It is a command-line program that - unlike "testRTSPClient" - is intended to be used as a complete, full-featured application (rather than having its code used within other applications). For more information about "openRTSP" - including its many command-line options - see the online documentation.

RTSP server

- **testOnDemandRTSPServer** creates a RTSP server that can stream, via RTP unicast, from various types of media file, on demand. (Supported media types include: MPEG-1 or 2 audio or video (elementary stream), including MP3 audio; MPEG-4 video (elementary stream); H.264 video (elementary stream); H.265 video (elementary stream); MPEG Program or Transport streams, including VOB files; DV video; AMR audio; WAV (PCM) audio.) The server can also stream from a Matroska or WebM file (by demultiplexing and streaming the tracks within the file). MPEG Transport Streams can also be streamed over raw UDP, if requested - e.g., by a set-top box.
  - This server application also demonstrates how to deliver - via RTSP - a MPEG Transport Stream that arrived at the server as a UDP (raw-UDP or RTP/UDP) multicast or unicast stream. In particular, it is set up, by default, to accept input from the "testMPEG2TransportStreamer" demo application.

SIP client

- **playSIP** is a command-line program (similar to "openRTSP") that makes a call to a SIP session (using a *sip:* URL), and then (optionally) records the incoming media stream into a file.

MP3 audio test programs

- **testMP3Streamer** repeatedly reads from a MP3 audio file (named "test.mp3"), and streams it, using RTP, to the multicast group 239.255.42.42, port 6666 (with RTCP using port 6667). This program also has an (optional) built-in RTSP server.
- **testMP3Receiver** does the reverse: It reads a MP3/RTP stream (from the same multicast group/port), and outputs the reconstituted MP3 stream to "stdout". It also sends RTCP Reception Reports.
  - Alternatively, the MP3/RTP stream could be played using one of these tools.

MPEG video test programs

- **testMPEG1or2VideoStreamer** repeatedly reads from a MPEG-1 or 2 video file (named "test.mpg"), and streams it, using RTP, to the multicast group 239.255.42.42, port 8888 (with RTCP using port 8889). This program also has an (optional) built-in RTSP server.
  - By default, the input file is assumed to be a MPEG Video Elementary Stream. If, however, it is a MPEG Program Stream, then you can also insert a demultiplexing filter to extract the Video Elementary Stream. (See "testMPEG1or2VideoStreamer.cpp" for details.)
  - Apple's "QuickTime Player" can be used to receive and view this streamed video (provided that it's MPEG-1, not MPEG-2). To use this, have QuickTime Player open the file "testMPEG1or2Video.sdp". (If "testMPEG1or2VideoStreamer's" RTSP server has been enabled, then QuickTime Player can also play the stream using a "rtsp://" URL.)
  - The Open Source "VLC" and "MPlayer" media players can also be used.
  - RealNetworks' "RealPlayer" can also be used to play the stream. (A recent version is recommended.)
- **testMPEG1or2VideoReceiver** does the reverse: It reads a MPEG Video/RTP stream (from the same multicast group/port), and outputs the reconstituted MPEG video (elementary) stream to "stdout". It also sends RTCP Reception Reports.
- **testMPEG4VideoStreamer** repeatedly reads from a MPEG-4 Elementary Stream video file (named "test.m4e"), and streams it using RTP multicast. This program also has a built-in RTSP server.
  - Apple's "QuickTime Player" can be used to receive and play this audio stream. To use this, have the player open the session's "rtsp://" URL (which the program prints out as it starts streaming).
  - The Open Source "VLC" and "MPlayer" media players can also be used.
- **testH264VideoStreamer** repeatedly reads from a H.264 Elementary Stream video file (named "test.264"), and streams it using RTP multicast. This program also has a built-in RTSP server.
  - Apple's "QuickTime Player" can be used to receive and play this audio stream. To use this, have the player open the session's "rtsp://" URL (which the program prints out as it starts streaming).
  - The Open Source "VLC" and "MPlayer" media players can also be used.
- **testH265VideoStreamer** repeatedly reads from a H.265 Elementary Stream video file (named "test.265"), and streams it using RTP multicast. This program also has a built-in RTSP server.

MPEG audio+video (Program Stream) test programs

- **testMPEG1or2AudioVideoStreamer** reads a MPEG-1 or 2 Program Stream file (named "test.mpg"), extracts from this an audio and a video Elementary Stream, and streams these, using RTP, to the multicast group 239.255.42.42, port 6666/6667 (for the audio stream) and 8888/8889 (for the video stream). This program also has an (optional) built-in RTSP server.
  - Apple's "QuickTime Player" can be used to receive and view this streamed video (provided that it's MPEG-1, not MPEG-2). To use this, have QuickTime Player open the file "testMPEG1or2AudioVideo.sdp". (If "testMPEG1or2VideoStreamer's" RTSP server has been enabled, then QuickTime Player can also play the stream using a "rtsp://" URL.)
  - The Open Source "VLC" and "MPlayer" media players can also be used.
- **testMPEG1or2Splitter** reads a MPEG-1 or 2 Program Stream file (named "in.mpg"), and extracts from this an audio and a video Elementary Stream. These two Elementary Streams are written into files named "out_audio.mpg" and "out_video.mpg" respectively.

MPEG audio+video (Transport Stream) test programs

- **testMPEG2TransportStreamer** reads a MPEG Transport Stream file (named "test.ts"), and streams it, using RTP, to the multicast group 239.255.42.42, port 1234 (with RTCP using port 1235). This program also has an (optional) built-in RTSP server.
  - The Open Source "VLC" media player can be used to play this stream.
- **testMPEG2TransportReceiver** does the reverse: It reads a MPEG Transport/RTP stream (from the same multicast group/port), and outputs the reconstituted MPEG Transport Stream stream to "stdout". It also sends RTCP Reception Reports.
- **testMPEG1or2ProgramToTransportStream** reads a MPEG-1 or 2 Program Stream file (named "in.mpg"), and converts it to an equivalent MPEG Transport Stream file, named "out.ts".
- **testH264VideoToTransportStream** reads a H.264 Video Elementary Stream file (named "in.264"), and converts it to an equivalent MPEG Transport Stream file, named "out.ts".
- **testH265VideoToTransportStream** reads a H.265 Video Elementary Stream file (named "in.265"), and converts it to an equivalent MPEG Transport Stream file, named "out.ts".
- **testMPEG2TransportStreamSplitter** demultiplexes a Transport Stream file into a set of output files, one for each of its component tracks. (Note that this application reads from 'stdin'.)

PCM audio test program

- **testWAVAudioStreamer** reads from a WAV-format audio file (named "test.wav"), and streams the enclosed PCM audio stream via IP multicast, using a built-in RTSP server.
  - The program supports 8 or 16-bit PCM streams, mono or stereo, at any sampling frequency.
  - Apple's "QuickTime Player" can be used to receive and play this audio stream. To use this, have the player open the session's "rtsp://" URL (which the program prints out as it starts streaming).
  - Optionally, 16-bit PCM streams can be converted to 8-bit u-law format prior to streaming. (See "testWAVAudioStreamer.cpp" for instructions on how to do this.)
  - The Open Source "VLC" and "MPlayer" media players can also be used.

AMR audio test program

- **testAMRAudioStreamer** reads from a AMR-format audio file (named "test.amr") - as defined in RFC 3267, section 5 - and streams the enclosed audio stream via IP multicast, using a built-in RTSP server.
  - Apple's "QuickTime Player" can be used to receive and play this audio stream. To use this, have the player open the session's "rtsp://" URL (which the program prints out as it starts streaming).

DV video test program

- **testDVVideoStreamer** reads from a DV video file (named "test.dv"), and streams it via IP multicast, using a built-in RTSP server.
  - At present, we know of no widely-available media player client that can play this stream.

Matroska (or 'Webm') test programs

- **testMKVStreamer** reads from a 'Matroska' (or 'Webm') file (named "test.mkv"), and streams it via IP multicast, using a built-in RTSP server.
- **testMKVSplitter** reads from a 'Matroska' (or 'Webm') file, and demultiplexes it into separate output files - one for each track.

VOB (DVD) streaming test program

- **vobStreamer** reads one or more ".vob" files (e.g., from a DVD), extracts the audio and video streams, and transmits them using RTP multicast.

Support for server 'trick play' operations on MPEG Transport Stream files

- The applications **MPEG2TransportStreamIndexer** and **testMPEG2TransportStreamTrickPlay**

HLS ("HTTP Live Streaming") test programs

- **testH264VideoToHLSSegments** converts a H.264 (Elementary Stream) video file - named "in.264" - into a sequence of HLS ("HTTP Live Streaming") segments, plus a ".m3u8" file that can be accessed via a web browser.
  (Note also the **LIVE555 HLS Proxy**, which converts a RTSP stream into a set of HLS segments in real time.)

Miscellaneous test programs

- **testRelay** repeatedly reads from a UDP multicast socket, and retransmits ('relays') each packet's payload to a new (multicast or unicast) address and port.
- **testReplicator** is similar to **testRelay**, except that it replicates the input stream - using the "FrameReplicator" class - and retransmits one replica stream to another (multicast or unicast) address and port, while writing the other replica stream to a file.
- **sapWatch** reads and prints SDP/SAP announcements made on the default SDP/SAP directory (224.2.127.254/9875)
- **registerRTSPStream** sends a custom RTSP "REGISTER" command to a given RTSP client (or proxy server), asking it to stream from a given "rtsp://" URL.
- **mikeyParse** parses a Base64 string - that encodes a binary MIKEY (multimedia key management) message - and outputs a human-readable description of the MIKEY message. (The Base64 string could, for example, have been used in a SDP "a=key-mgmt:" attribute, as defined by RFC 4567.)

WindowsAudioInputDevice

This is an implementation of the "liveMedia" library's "AudioInputDevice" abstract class. This can be used by a Windows application to read PCM audio samples from an input device.

(This project builds two libraries: "libWindowsAudioInputDevice_mixer.lib", which uses Windows' built-in mixer, and "libWindowsAudioInputDevice_noMixer.lib", which doesn't.)

How to configure and build the code on Unix (including Linux, Mac OS X, QNX, and other Posix-compliant systems)

The source code package can be found (as a ".tar.gz" file) here. Use "tar -x" and "gunzip" (or "tar -xz", if available) to extract the package; then cd to the "live" directory. Then run

```
./genMakefiles <os-platform>
```

where <*os-platform*> is your target platform - e.g., "linux" or "solaris" - defined by a "config.<*os-platform*>" file. This will generate a Makefile in the "live" directory and each subdirectory. Then run "make".

- If the "make" fails, you may need to make small modifications to the appropriate "config.<*os-platform*>" file, and then re-run "genMakefiles <*os-platform*>". (E.g., you may need to add another "-I<*dir*>" flag to the COMPILE_OPTS definition.)
- Some people (in particular, FreeBSD users) have reported that the GNU version of "make" - often called "gmake" - works better than their default, pre-installed version of "make". (In particular, you should try using "gmake" if you encounter linking problems with the "ar" command.)
- **If you're using "gcc" version 3.0 or greater:** You may also wish to add the *-Wno-deprecated* flag to CPLUSPLUS_FLAGS.
- If no "config.<*os-platform*>" file exists for your target platform, then try using one of the existing files as a template.

If you wish, you can also 'install' the headers, libraries, and applications by running "make install".

How to configure and build the code on Windows

The use of Windows is not recommended. However, if you insist on using Windows, you might be able to build the code in the "Cygwin" environment by doing:

```
cd live
./genMakefiles cygwin
make
```

Frequently Asked Questions (FAQ)

(Please read this before posting questions to the mailing list.)

Source code license

This code is "open source", and is released under the LGPL. This allows you to use these libraries (via linking) inside closed-source products. It also allows you to make closed-source binary extensions to these libraries - for instance, to support proprietary media codecs that subclass the existing "liveMedia" class hierarchy. Nonetheless, we hope that subclass extensions of these libraries will also be developed under the LGPL, and contributed for inclusion here. We encourage developers to contribute to the development and enhancement of these libraries.

To do...

- Extend the "liveMedia" library to support even more media types and/or codecs.
- Supply additional "UsageEnvironment" subclassed implementations - e.g., for use with scripted environments, such as Tcl, Python, or Perl.
- Expand the RTCP implementation to support more SDES items (other than just CNAME), and to more completely handle incoming RTCP packets.
- In places the code uses data types such as "unsigned", where instead specific lengths (e.g., 32 bits) are required. The data types should be changed in each such case.
- Fix the "groupsock" implementation so that it could use IPv6 instead of IPv4. (At present, 4-byte addresses are hard-wired into the code in several places.)
- Switch to using ISO-conformant C++ headers (but in such a way that the code can remain portable across both Unix (using gcc) and Windows (using Visual C++)).
- Implement network sockets ("groupsock"s) as "liveMedia" 'sources'. This will make the code for RTP sources more consistent with the rest of the "liveMedia" library, allowing them to be "FramedFilters". (This will make it possible to use synthetic network sockets - e.g., for debugging or simulation.)
- Add support for SRTP ('secure' RTP), and perhaps also RTSP-over-TLS.

Some third-party applications

Several third-party devices and applications have made use of the LIVE555 Streaming Media Software. Here is a list of some of them:

Hardware

- "RaspberrIPCam" - a full HD IP Camera based on Raspberry Pi
- Network-enabled, RTP-streaming Security Cameras, from RVision
- The "LMLM4" MPEG-4 audio/video hardware encoder PCI board, from Linux Media Labs.
- WIS Technologies media encoders. (The "wis-streamer" server application - for Linux - can be used to stream from many of these encoders.)
- Elphel Network Video Cameras (with optional JPEG streaming over RTP).
- The "Cam.Jy" wireless security camera system uses "openRTSP" to archive recorded video.

Software

- The "VLC" media player (uses the "LIVE555 Streaming Media" libraries for its RTSP client implementation).
- The "MPlayer media player.
- Ralf Globisch's "Video Processing Project" - a RTSP client DirectShow Filter
- "Morgan RTP DirectShow™ Filters", from Morgan Multimedia
- "Omnimeeting" - a videoconferencing application
- "Network-Integrated Multimedia Middleware (NMM)", from Saarland University
- "ivrworx" - a SIP IVR application - uses "LIVE555 Streaming Media" code for RTP mixing
- "LiveMediaStreamer" - an open source multimedia framework
- "MediaPortal - an open source media center
- "Valkka" - an open source video management library

If you would like your product (or project) added to this list, please send email to the developers' mailing list.