HTTP Live Streaming Overview

Developer

Contents

Figures, Tables, and Listings

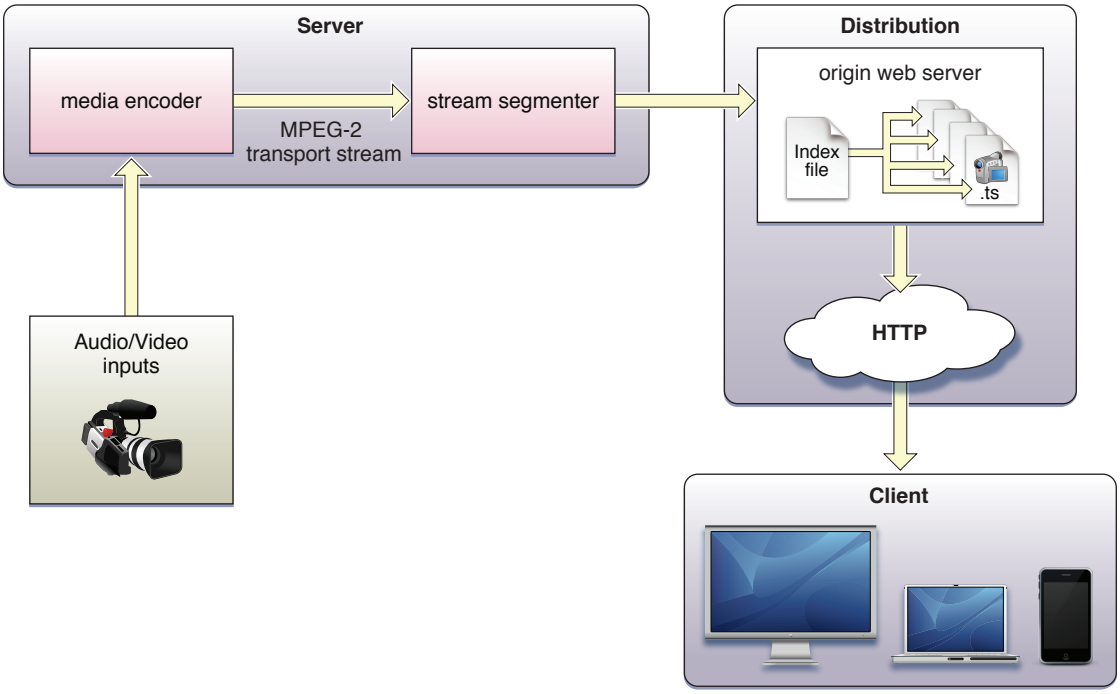
Introduction

If you are interested in any of the following:

- Streaming audio or video to iPhone, iPod touch, iPad, or Apple TV
- Streaming live events without special server software
- Sending video on demand with encryption and authentication

you should learn about HTTP Live Streaming.

HTTP Live Streaming lets you send audio and video over HTTP from an ordinary web server for playback on iOS-based devices—including iPhone, iPad, iPod touch, and Apple TV—and on desktop computers (Mac OS X). HTTP Live Streaming supports both live broadcasts and prerecorded content (video on demand). HTTP Live Streaming supports multiple alternate streams at different bit rates, and the client software can switch streams intelligently as network bandwidth changes. HTTP Live Streaming also provides for media encryption and user authentication over HTTPS, allowing publishers to protect their work.



All devices running iOS 3.0 and later include built-in client software for HTTP Live Streaming. The Safari browser can play HTTP streams within a webpage on iPad and desktop computers, and Safari launches a full-screen media player for HTTP streams on iOS devices with small screens, such as iPhone and iPod touch. Apple TV 2 and later includes an HTTP Live Streaming client.

Important: iPhone and iPad apps that send large amounts of audio or video data over cellular networks are *required* to use HTTP Live Streaming. See "Requirements for Apps" (page 20).

Safari plays HTTP Live streams natively as the source for the `<video>` tag. Mac OS X developers can use the QTKit and AVFoundation frameworks to create desktop applications that play HTTP Live Streams, and iOS developers can use the MediaPlayer and AVFoundation frameworks to create iOS apps.

Important: Where possible, use the `<video>` tag to embed HTTP Live Streaming, and use the `<object>` or `<embed>` tags only to specify fallback content.

Because it uses HTTP, this kind of streaming is automatically supported by nearly all edge servers, media distributors, caching systems, routers, and firewalls.

Note: Many existing streaming services require specialized servers to distribute content to end users. These servers require specialized skills to set up and maintain, and in a large-scale deployment this can be costly. HTTP Live Streaming avoids this by using standard HTTP to deliver the media. Additionally, HTTP Live Streaming is designed to work seamlessly in conjunction with media distribution networks for large scale operations.

The HTTP Live Streaming specification is an IETF Internet-Draft. For a link to the specification, see the See Also section below.

At a Glance

HTTP Live Streaming is a way to send audio and video over HTTP from a web server to client software on the desktop or to iOS-based devices.

You Can Send Audio and Video Without Special Server Software

You can serve HTTP Live Streaming audio and video from an ordinary web server. The client software can be the Safari browser or an app that you've written for iOS or Mac OS X.

HTTP Live Streaming sends audio and video as a series of small files, typically of about 10 seconds duration, called media segment files. An index file, or playlist, gives the clients the URLs of the media segment files. The playlist can be periodically refreshed to accommodate live broadcasts, where media segment files are constantly being produced. You can embed a link to the playlist in a webpage or send it to an app that you've written.

Relevant Chapter: "HTTP Streaming Architecture" (page 8)

You Can Send Live Streams or Video on Demand, with Optional Encryption

For video on demand from prerecorded media, Apple provides a free tool to make media segment files and playlists from MPEG-4 video or QuickTime movies with H.264 video compression, or audio files with AAC or MP3 compression. The playlists and media segment files can be used for video on demand or streaming radio, for example.

For live streams, Apple provides a free tool to make media segment files and playlists from live MPEG-2 transport streams carrying H.264 video, AAC audio, or MP3 audio. There are a number of hardware and software encoders that can create MPEG-2 transport streams carrying MPEG-4 video and AAC audio in real time.

These tools can be instructed to encrypt your media and generate decryption keys. You can use a single key for all your streams, a different key for each stream, or a series of randomly generated keys that change at intervals during a stream. Keys are further protected by the requirement for an initialization vector, which can also be set to change periodically.

Relevant Chapter: "Using HTTP Live Streaming" (page 14)

Prerequisites

You should have a general understanding of common audio and video file formats and be familiar with how web servers and browsers work.

See Also

- *iOS Human Interface Guidelines* —how to design web content for iOS-based devices.
- HTTP Live Streaming protocol—the IETF Internet-Draft of the HTTP Live Streaming specification.
- HTTP Live Streaming Resources—a collection of information and tools to help you get started.

HTTP Streaming Architecture

HTTP Live Streaming allows you to send live or prerecorded audio and video, with support for encryption and authentication, from an ordinary web server to any device running iOS 3.0 or later (including iPad and Apple TV), or any computer with Safari 4.0 or later installed.

Overview

Conceptually, HTTP Live Streaming consists of three parts: the server component, the distribution component, and the client software.

The **server component** is responsible for taking input streams of media and encoding them digitally, encapsulating them in a format suitable for delivery, and preparing the encapsulated media for distribution.

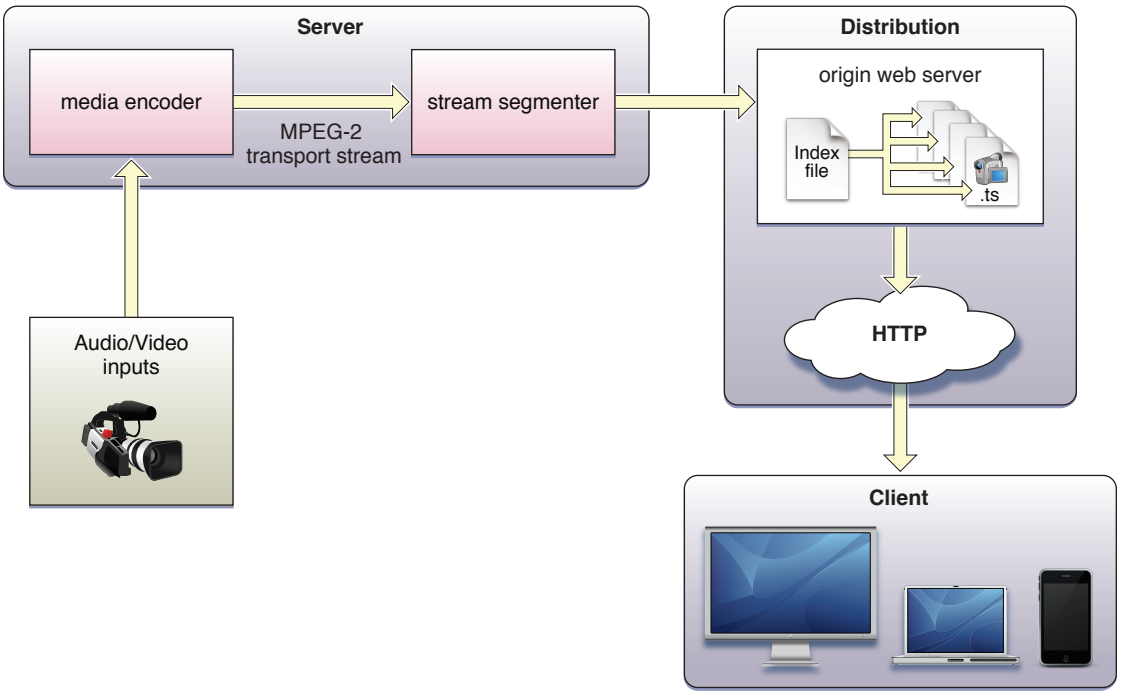
The **distribution component** consists of standard web servers. They are responsible for accepting client requests and delivering prepared media and associated resources to the client. For large-scale distribution, edge networks or other content delivery networks can also be used.

The **client software** is responsible for determining the appropriate media to request, downloading those resources, and then reassembling them so that the media can be presented to the user in a continuous stream. Client software is included on iOS 3.0 and later and computers with Safari 4.0 or later installed.

In a typical configuration, a hardware encoder takes audio-video input, encodes it as H.264 video and AAC audio, and outputs it in an MPEG-2 Transport Stream, which is then broken into a series of short media files by a software stream segmenter. These files are placed on a web server. The segmenter also creates and maintains an index file containing a list of the media files. The URL of the index file is published on the web server. Client software reads the index, then requests the listed media files in order and displays them without any pauses or gaps between segments.

An example of a simple HTTP streaming configuration is shown in Figure 1-1.

Figure 1-1 A basic configuration



Input can be live or from a prerecorded source. It is typically encoded as MPEG-4 (H.264 video and AAC audio) and packaged in an MPEG-2 Transport Stream by off-the-shelf hardware. The MPEG-2 transport stream is then broken into segments and saved as a series of one or more `.ts` media files. This is typically accomplished using a software tool such as the Apple stream segmenter.

Audio-only streams can be a series of MPEG elementary audio files formatted as AAC with ADTS headers, as MP3, or as AC-3.

The segmenter also creates an index file. The index file contains a list of media files. The index file also contains metadata. The index file is an `.M3U8` playlist. The URL of the index file is accessed by clients, which then request the indexed files in sequence.

Server Components

The server requires a media encoder, which can be off-the-shelf hardware, and a way to break the encoded media into segments and save them as files, which can either be software such as the media stream segmenter provided by Apple or part of an integrated third-party solution.

Media Encoder

The media encoder takes a real-time signal from an audio-video device, encodes the media, and encapsulates it for transport. Encoding should be set to a format supported by the client device, such as H.264 video and HE-AAC audio. Currently, the supported delivery format is MPEG-2 Transport Streams for audio-video, or MPEG elementary streams for audio-only.

The encoder delivers the encoded media in an MPEG-2 Transport Stream over the local network to the stream segmenter.

> **Note:** MPEG-2 transport streams should not be confused with MPEG-2 video compression. The transport stream is a packaging format that can be used with a number of different compression formats. Only MPEG-2 transport streams with H.264 video and AAC, AC-3 or MP3 audio are supported at this time for audio-video content. Audio-only content can be either MPEG-2 transport or MPEG elementary audio streams, and can be in AAC format with ADTS headers, in MP3 format, or in AC-3 format.

Important: The video encoder should not change stream settings—such as video dimensions or codec type—in the midst of encoding a stream. If a stream settings change is unavoidable, the settings must change at a segment boundary, and the `EXT-X-DISCONTINUITY` tag must be set on the following segment.

Stream Segmenter

The stream segmenter is a process—typically software—that reads the Transport Stream from the local network and divides it into a series of small media files of equal duration. Even though each segment is in a separate file, video files are made from a continuous stream which can be reconstructed seamlessly.

The segmenter also creates an index file containing references to the individual media files. Each time the segmenter completes a new media file, the index file is updated. The index is used to track the availability and location of the media files. The segmenter may also encrypt each media segment and create a key file as part of the process.

Media segments are saved as `.ts` files (MPEG-2 transport stream files). Index files are saved as `.M3U8` playlists.

File Segmenter

If you already have a media file encoded using supported codecs, you can use a file segmenter to encapsulate it in an MPEG-2 transport stream and break it into segments of equal length. The file segmenter allows you to use a library of existing audio and video files for sending video on demand via HTTP Live Streaming. The file segmenter performs the same tasks as the stream segmenter, but it takes files as input instead of streams.

Media Segment Files

The media segment files are normally produced by the stream segmenter, based on input from the encoder, and consist of a series of `.ts` files containing segments of an MPEG-2 Transport Stream carrying H.264 video and AAC, MP3, or AC-3 audio. For an audio-only broadcast, the segmenter can produce MPEG elementary audio streams containing either AAC audio with ADTS headers, MP3 audio, or AC-3 audio.

Index Files (Playlists)

Index files are normally produced by the stream segmenter or file segmenter, and saved as `.M3U8` playlists, an extension of the `.m3u` format used for MP3 playlists.

> **Note:** Because the index file format is an extension of the `.m3u` playlist format, and because the system also supports `.mp3` audio media files, the client software may also be compatible with typical MP3 playlists used for streaming Internet radio.

Here is a very simple example of an index file, in the form of an `.M3U8` playlist, that a segmenter might produce if the entire stream were contained in three unencrypted 10-second media files:

```
#EXT-X-VERSION:3
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1

# Old-style integer duration; avoid for newer clients.
#EXTINF:10,
http://media.example.com/segment0.ts

# New-style floating-point duration; use for modern clients.
#EXTINF:10.0,
http://media.example.com/segment1.ts
#EXTINF:9.5,
http://media.example.com/segment2.ts
#EXT-X-ENDLIST
```

For maximum accuracy, you should specify all durations as floating-point values when sending playlists to clients that support version 3 of the protocol or later. (Older clients support only integer values.) You must specify a protocol version when using floating-point lengths; if the version is omitted, the playlist must conform to version 1 of the protocol.

> **Note:** You can use the file segmenter provided by Apple to generate a variety of example playlists, using an MPEG-4 video or AAC or MP3 audio file as a source. For details, see "Media File Segmenter" (page 14).

The index file may also contain URLs for encryption key files and alternate index files for different bandwidths. For details of the index file format, see the IETF Internet-Draft of the HTTP Live Streaming specification.

Index files are normally created by the same segmenter that creates the media segment files. Alternatively, it is possible to create the `.M3U8` file and the media segment files independently, provided they conform the published specification. For audio-only broadcasts, for example, you could create an `.M3U8` file using a text editor, listing a series of existing `.MP3` files.

Distribution Components

The distribution system is a web server or a web caching system that delivers the media files and index files to the client over HTTP. No custom server modules are required to deliver the content, and typically very little configuration is needed on the web server.

Recommended configuration is typically limited to specifying MIME-type associations for `.M3U8` files and `.ts` files.

For details, see "Deploying HTTP Live Streaming" (page 26).

Client Component

The client software begins by fetching the index file, based on a URL identifying the stream. The index file in turn specifies the location of the available media files, decryption keys, and any alternate streams available. For the selected stream, the client downloads each available media file in sequence. Each file contains a consecutive segment of the stream. Once it has a sufficient amount of data downloaded, the client begins presenting the reassembled stream to the user.

The client is responsible for fetching any decryption keys, authenticating or presenting a user interface to allow authentication, and decrypting media files as needed.

This process continues until the client encounters the `#EXT-X-ENDLIST` tag in the index file. If no `#EXT-X-ENDLIST` tag is present, the index file is part of an ongoing broadcast. During ongoing broadcasts, the client loads a new version of the index file periodically. The client looks for new media files and encryption keys in the updated index and adds these URLs to its queue.

Using HTTP Live Streaming

Download the Tools

There are several tools available that can help you set up an HTTP Live Streaming service. The tools include a media stream segmenter, a media file segmenter, a stream validator, an id3 tag generator, and a variant playlist generator.

The tools are frequently updated, so you should download the current version of the HTTP Live Streaming Tools from the Apple Developer website. You can access them if you are a member of the iOS Developer Program. One way to navigate to the tools is to log onto developer.apple.com, then use the search feature.

Media Stream Segmenter

The `mediastreamsegmenter` command-line tool takes an MPEG-2 transport stream as an input and produces a series of equal-length files from it, suitable for use in HTTP Live Streaming. It can also generate index files (also known as playlists), encrypt the media, produce encryption keys, optimize the files by reducing overhead, and create the necessary files for automatically generating multiple stream alternates. For details, verify you have installed the tools, and type `man mediastreamsegmenter` from the terminal window.

Usage example: `mediastreamsegmenter -s 3 -D -f /Library/WebServer/Documents/stream 239.4.1.5:20103`

The usage example captures a live stream from the network at address `239.4.1.5:20103` and creates media segment files and index files from it. The index files contain a list of the current three media segment files (`-s 3`). The media segment files are deleted after use (`-D`). The index files and media segment files are stored in the directory `/Library/WebServer/Documents/stream`.

Media File Segmenter

The `mediafilesegmenter` command-line tool takes an encoded media file as an input, wraps it in an MPEG-2 transport stream, and produces a series of equal-length files from it, suitable for use in HTTP Live Streaming. The media file segmenter can also produce index files (playlists) and decryption keys. The file segmenter behaves very much like the stream segmenter, but it works on existing files instead of streams coming from an encoder. For details, type `man mediafilesegmenter` from the terminal window.

Media Stream Validator

The `mediastreamvalidator` command-line tool examines the index files, stream alternates, and media segment files on a server and tests to determine whether they will work with HTTP Live Streaming clients. For details, type `man mediastreamvalidator` from the terminal window.

Variant Playlist Creator

The `variantplaylistcreator` command-line tool creates a master index file, or playlist, listing the index files for alternate streams at different bit rates, using the output of the `mediafilesegmenter`. The `mediafilesegmenter` must be invoked with the `–generate–variant–playlist` argument to produce the required output for the variant playlist creator. For details, type `man variantplaylistcreator` from the terminal window.

Metadata Tag Generator

The `id3taggenerator` command-line tool generates ID3 metadata tags. These tags can either be written to a file or inserted into outgoing stream segments. For details, see "Adding Timed Metadata" (page 21).

Session Types

The HTTP Live Streaming protocol supports two types of sessions: events (live broadcasts) and video on demand (VOD).

VOD Sessions

For VOD sessions, media files are available representing the entire duration of the presentation. The index file is static and contains a complete list of all files created since the beginning of the presentation. This kind of session allows the client full access to the entire program.

VOD can also be used to deliver "canned" media. HTTP Live Streaming offers advantages over progressive download for VOD, such as support for media encryption and dynamic switching between streams of different data rates in response to changing connection speeds. (QuickTime also supports multiple-data-rate movies using progressive download, but QuickTime movies do not support dynamically switching between data rates in mid-movie.)

Live Sessions

Live sessions (events) can be presented as a complete record of an event, or as a sliding window with a limited time range the user can seek within.

For live sessions, as new media files are created and made available, the index file is updated. The new index file lists the new media files. Older media files can be removed from the index and discarded, presenting a moving window into a continuous stream—this type of session is suitable for continuous broadcasts. Alternatively, the index can simply add new media files to the existing list—this type of session can be easily converted to VOD after the event completes.

It is possible to create a live broadcast of an event that is instantly available for video on demand. To convert a live broadcast to VOD, do not remove the old media files from the server or delete their URLs from the index file; instead, add an `#EXT-X-ENDLIST` tag to the index when the event ends. This allows clients to join the broadcast late and still see the entire event. It also allows an event to be archived for rebroadcast with no additional time or effort.

If your playlist contains an `EXT-X-PLAYLIST-TYPE` tag, you should also change the value from `EVENT` to `VOD`.

Content Protection

Media files containing stream segments may be individually encrypted. When encryption is employed, references to the corresponding key files appear in the index file so that the client can retrieve the keys for decryption.

When a key file is listed in the index file, the key file contains a cipher key that must be used to decrypt subsequent media files listed in the index file. Currently HTTP Live Streaming supports AES-128 encryption using 16-octet keys. The format of the key file is a packed array of these 16 octets in binary format.

The media stream segmenter available from Apple provides encryption and supports three modes for configuring encryption.

The first mode allows you to specify a path to an existing key file on disk. In this mode the segmenter inserts the URL of the existing key file in the index file. It encrypts all media files using this key.

The second mode instructs the segmenter to generate a random key file, save it in a specified location, and reference it in the index file. All media files are encrypted using this randomly generated key.

The third mode instructs the segmenter to generate a new random key file every *n* media segments, save it in a specified location, and reference it in the index file. This mode is referred to as key rotation. Each group of *n* files is encrypted using a different key.

Note: All media files may be encrypted using the same key, or new keys may be required at intervals. The theoretical limit is one key per media file, but because each media key adds a file request and transfer to the overhead for presenting the subsequent media segments, changing to a new key periodically is less likely to impact system performance than changing keys for each segment.

You can serve key files using either HTTP or HTTPS. You may also choose to protect the delivery of the key files using your own session-based authentication scheme. For details, see "Serving Key Files Securely Over HTTPS" (page 28).

Key files require an initialization vector (IV) to decode encrypted media. The IVs can be changed periodically, just as the keys can.

Caching and Delivery Protocols

HTTPS is commonly used to deliver key files. It may also be used to deliver the media segment files and index files, but this is not recommended when scalability is important, since HTTPS requests often bypass web server caches, causing all content requests to be routed through your server and defeating the purpose of edge network distribution systems.

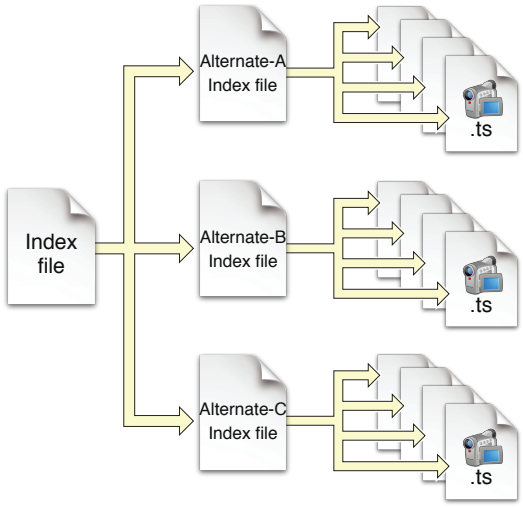
For this very reason, however, it is important to make sure that any content delivery network you use understands that the `.M3U8` index files are not to be cached for longer than one media segment duration for live broadcasts, where the index file is changing dynamically.

Stream Alternates

A master index file may reference alternate streams of content. References can be used to support delivery of multiple streams of the same content with varying quality levels for different bandwidths or devices. HTTP Live Streaming supports switching between streams dynamically if the available bandwidth changes. The client software uses heuristics to determine appropriate times to switch between the alternates. Currently, these heuristics are based on recent trends in measured network throughput.

The master index file points to alternate streams of media by including a specially tagged list of other index files, as illustrated in Figure 2-1

Figure 2-1 Stream alternates



Both the master index file and the alternate index files are in `.M3U8` playlist format. The master index file is downloaded only once, but for live broadcasts the alternate index files are reloaded periodically. The first alternate listed in the master index file is the first stream used—after that, the client chooses among the alternates by available bandwidth.

Note that the client may choose to change to an alternate stream at any time, such as when a mobile device enters or leaves a WiFi hotspot. All alternates should use identical audio to allow smooth transitions among streams.

You can create a set of stream alternates by using the `variantplaylistcreator` tool and specifying the `–generate–variant–playlist` option for either the `mediafilesegmenter` tool or the `mediastreamsegmenter` tool (see "Download the Tools" (page 14) for details).

When using stream alternates, it is important to bear the following considerations in mind:

- The first entry in the variant playlist is played when a user joins the stream and is used as part of a test to determine which stream is most appropriate. The order of the other entries is irrelevant.
- Where possible, encode enough variants to provide the best quality stream across a wide range of connection speeds. For example, encode variants at 150 kbps, 350 kbps, 550 kbps, 900 kbps, 1500 kbps.
- When possible, use relative path names in variant playlists and in the individual `.M3U8` playlist files
- The video aspect ratio on alternate streams *must* be exactly the same, but alternates can have different pixel dimensions, as long as they have the same aspect ratio. For example, two stream alternates with the same 4:3 aspect ratio could have dimensions of 400 x 300 and 800 x 600.

- The `RESOLUTION` field in the `EXT-X-STREAM-INF` should be included to help the client choose an appropriate stream.
- If you are an iOS app developer, you can query the user's device to determine whether the initial connection is cellular or WiFi and choose an appropriate master index file.

  To ensure the user has a good experience when the stream is first played, regardless of the initial network connection, you should have more than one master index file consisting of the same alternate index files but with a different first stream.

  A 150k stream for the cellular variant playlist is recommended.

  A 240k or 440k stream for the Wi-Fi variant playlist is recommended.

  > **Note:** For details on how to query an iOS-based device for its network connection type, see the following sample code: *Reachability*.

- When you specify the bitrates for stream variants, it is important that the `BANDWIDTH` attribute closely match the actual bandwidth required by a given stream. If the actual bandwidth requirement is substantially different than the `BANDWIDTH` attribute, automatic switching of streams may not operate smoothly or even correctly.

Video Over Cellular Networks

When you send video to a mobile device such as iPhone or iPad, the client's Internet connection may move to or from a cellular network at any time.

HTTP Live Streaming allows the client to choose among stream alternates dynamically as the network bandwidth changes, providing the best stream as the device moves between cellular and WiFi connections, for example, or between 3G and EDGE connections. This is a significant advantage over progressive download.

It is strongly recommended that you use HTTP Live Streaming to deliver video to all cellular-capable devices, even for video on demand, so that your viewers have the best experience possible under changing conditions.

In addition, you should provide cellular-capable clients an alternate stream at 64 Kbps or less for slower data connections. If you cannot provide video of acceptable quality at 64 Kbps or lower, you should provide an audio-only stream, or audio with a still image.

A good choice for pixel dimensions when targeting cellular network connections is 400 x 224 for 16:9 content and 400 x 300 for 4:3 content (see "Preparing Media for Delivery to iOS-Based Devices" (page 23)).

Requirements for Apps

Warning: iOS apps submitted for distribution in the App Store must conform to these requirements.

If your app delivers video over cellular networks, and the video exceeds either 10 minutes duration or 5 MB of data in a five minute period, you are required to use HTTP Live Streaming. (Progressive download may be used for smaller clips.)

If your app uses HTTP Live Streaming over cellular networks, you are required to provide at least one stream at 64 Kbps or lower bandwidth (the low-bandwidth stream may be audio-only or audio with a still image).

These requirements apply to iOS apps submitted for distribution in the App Store for use on Apple products. Non-compliant apps may be rejected or removed, at the discretion of Apple.

Redundant Streams

If your playlist contains alternate streams, they can not only operate as bandwidth or device alternates, but as failure fallbacks. Starting with iOS 3.1, if the client is unable to reload the index file for a stream (due to a 404 error, for example), the client attempts to switch to an alternate stream.

In the event of an index load failure on one stream, the client chooses the highest bandwidth alternate stream that the network connection supports. If there are multiple alternates at the same bandwidth, the client chooses among them in the order listed in the playlist.

You can use this feature to provide redundant streams that will allow media to reach clients even in the event of severe local failures, such as a server crashing or a content distributor node going down.

To support redundant streams, create a stream—or multiple alternate bandwidth streams—and generate a playlist file as you normally would. Then create a parallel stream, or set of streams, on a separate server or content distribution service. Add the list of backup streams to the playlist file, so that the backup stream at each bandwidth is listed after the primary stream. For example, if the primary stream comes from server ALPHA, and the backup stream is on server BETA, your playlist file might look something like this:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000, RESOLUTION=720x480
http://ALPHA.mycompany.com/lo/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000, RESOLUTION=720x480
http://BETA.mycompany.com/lo/prog_index.m3u8
```

```
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000, RESOLUTION=1920x1080
http://ALPHA.mycompany.com/md/prog_index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000, RESOLUTION=1920x1080
http://BETA.mycompany.com/md/prog_index.m3u8
```

Note that the backup streams are intermixed with the primary streams in the playlist, with the backup at each bandwidth listed after the primary for that bandwidth.

You are not limited to a single backup stream set. In the example above, ALPHA and BETA could be followed by GAMMA, for instance. Similarly, you need not provide a complete parallel set of streams. You could provide a single low-bandwidth stream on a backup server, for example.

Adding Timed Metadata

You can add various kinds of metadata to media stream segments. For example, you can add the album art, artist's name, and song title to an audio stream. As another example, you could add the current batter's name and statistics to video of a baseball game.

If an audio-only stream includes an image as metadata, the Apple client software automatically displays it. Currently, the only metadata that is automatically displayed by the Apple-supplied client software is a still image accompanying an audio-only stream.

If you are writing your own client software, however, using either `MPMoviePlayerController` or `AVPlayerItem`, you can access streamed metadata using the `timedMetaData` property.

If you are writing your own segmenter, you can read about the stream format for timed metadata information in *Timed Metadata for HTTP Live Streaming*.

If you are using Apple's tools, you can add timed metadata by specifying a metadata file in the `-F` command line option to either the stream segmenter or the file segmenter. The specified metadata source can be a file in ID3 format or an image file (JPEG or PNG). Metadata specified this way is automatically inserted into every media segment.

This is called timed metadata because it is inserted into a media stream at a given time offset. Timed metadata can optionally be inserted into all segments after a given time.

To add timed metadata to a live stream, use the `id3taggenerator` tool, with its output set to the stream segmenter. The tool generates ID3 metadata and passes it the stream segmenter for inclusion in the outbound stream.

The tag generator can be run from a shell script, for example, to insert metadata at the desired time, or at desired intervals. New timed metadata automatically replaces any existing metadata.

Once metadata has been inserted into a media segment, it is persistent. If a live broadcast is re-purposed as video on demand, for example, it retains any metadata inserted during the original broadcast.

Adding timed metadata to a stream created using the file segmenter is slightly more complicated.

1. First, generate the metadata samples. You can generate ID3 metadata using the `id3taggenerator` command-line tool, with the output set to file.
2. Next, create a **metadata macro file**—a text file in which each line contains the time to insert the metadata, the type of metadata, and the path and filename of a metadata file.

   For example, the following metadata macro file would insert a picture at 1.2 seconds into the stream, then an ID3 tag at 10 seconds:

   ```
   1.2 picture /meta/images/picture.jpg
   10 id3 /meta/id3/title.id3
   ```
3. Finally, specify the metadata macro file by name when you invoke the media file segmenter, using the `–M` command line option.

For additional details, see the `man` pages for `mediastreamsegmenter`, `mediafilesegmenter`, and `id3taggenerator`.

Adding Closed Captions

HTTP Live Streaming supports adding closed captions to streams.

> **Note:** Closed captions should not be confused with subtitles.

If you are using the stream segmenter, you need to add CEA-608 closed captions to the MPEG-2 transport stream (in the main video elementary stream) as specified in ATSC A/72.

If you are using the file segmenter, you should encapsulate your media in a QuickTime movie file and add a closed caption track (`'clcp'`).

One tool you can use to add closed captions to movie files or MPEG-2 transport streams is Compressor, included with Final Cut Studio. Compressor works in conjunction with Scenarist software to generate closed captions. See the Compressor documentation for details.

If you are writing an app, the AVFoundation framework supports playback of closed captions.

Preparing Media for Delivery to iOS-Based Devices

The recommended encoder settings for streams used with iOS-based devices are shown in the following four tables. For live streams, these settings should be available from your hardware or software encoder. If you are re-encoding from a master file for video on demand, you can use a video editing tool such as Compressor.

File format for the file segmenter can be a QuickTime movie, MPEG-4 video, or MP3 audio, using the specified encoding.

Stream format for the stream segmenter must be MPEG elementary audio and video streams, wrapped in an MPEG-2 transport stream, and using the following encoding.

- Encode video using H.264 compression
    - H.264 Baseline 3.0: All devices
    - H.264 Baseline 3.1: iPhone 3G and later, and iPod touch 2nd generation and later.
    - H.264 Main profile 3.1: iPad (all versions), Apple TV 2 and later, and iPhone 4 and later.
    - H.264 Main Profile 4.0: Apple TV 3 and later, iPad 2 and later, and iPhone 4S and later
    - H.264 High Profile 4.0: Apple TV 3 and later, iPad 2 and later, and iPhone 4S and later.
    - H.264 High Profile 4.1: iPad 2 and later and iPhone 4S and later.
- A frame rate of 10 fps is recommended for video streams under 200 kbps. For video streams under 300 kbps, a frame rate of 12 to 15 fps is recommended. For all other streams, a frame rate of 29.97 is recommended.
- Encode audio as either of the following:
    - HE-AAC or AAC-LC, stereo
    - MP3 (MPEG-1 Audio Layer 3), stereo
- Optionally include an AC-3 audio track for Apple TV (or AirPlay to an Apple TV) when used with a surround sound receiver or TV that supports AC-3 input.
- A minimum audio sample rate of 22.05 kHz and audio bit rate of 40 kbps is recommended in all cases, and higher sampling rates and bit rates are strongly encouraged when streaming over Wi-Fi.

Table 2-1 Baseline profile encoder settings, 16:9 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
Cellular	400 x 224	64 kbps	audio only	none
Cellular	400 x 224	150 kbps	110 kbps	30
Cellular	400 x 224	240 kbps	200 kbps	45
Cellular	400 x 224	440 kbps	400 kbps	90
WiFi	640 x 360	640 kbps	600 kbps	90

Table 2-2 Baseline profile encoder settings, 4:3 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
Cellular	400 x 300	64 kbps	audio only	none
Cellular	400 x 300	150 kbps	110 kbps	30
Cellular	400 x 300	240 kbps	200 kbps	45
Cellular	400 x 300	440 kbps	400 kbps	90
WiFi	640 x 480	640 kbps	600 kbps	90

Table 2-3 Additional main profile encoder settings, 16:9 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
WiFi	640 x 360	1240 kbps	1200 kbps	90
WiFi	960 x 540	1840 kbps	1800 kbps	90
WiFi	1280 x 720	2540 kbps	1500 kbps	90
WiFi	1280 x 720	4540 kbps	4500 kbps	90

Table 2-4 Additional main profile encoder settings, 4:3 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
WiFi	640 x 480	1240 kbps	1200 kbps	90
WiFi	960 x 720	1840 kbps	1800 kbps	90
WiFi	960 x 720	2540 kbps	2500 kbps	90
WiFi	1280 x 960	4540 kbps	4500 kbps	90

Table 2-5 Additional high profile encoder settings, 16:9 aspect ratio

Connection	Dimensions	Total bit rate	Video bit rate	Keyframes
WiFi	1920 x 1080	12000 kbps	11000 kbps	90
WiFi	1920 x 1080	25000 kbps	24000 kbps	90
WiFi	1920 x 1080	40000 kbps	39000 kbps	90

Sample Streams

There are a series of HTTP streams available for testing on Apple's developer site. These examples show proper formatting of HTML to embed streams, `.M3U8` files to index the streams, and `.ts` media segment files. The streams can be accessed from the HTTP Live Streaming Resources.

Deploying HTTP Live Streaming

To actually deploy HTTP Live Streaming, you need to create either an HTML page for browsers or a client app to act as a receiver. You also need the use of a web server and a way to either encode live streams as MPEG-2 transport streams or to create MP3 or MPEG-4 media files with H.264 and AAC encoding from your source material.

You can use the Apple-provided tools to segment the streams or media files, and to produce the index files and variant playlists (see "Download the Tools" (page 14)).

You should use the Apple-provided media stream validator prior to serving your streams, to ensure that they are fully compliant with HTTP Live Streaming.

You may want to encrypt your streams, in which case you probably also want to serve the encryption key files securely over HTTPS, so that only your intended clients can decrypt them.

Creating an HTML Page

The easiest way to distribute HTTP Live Streaming media is to create a webpage that includes the HTML5 `<video>` tag, using an `.M3U8` playlist file as the video source. An example is shown in Listing 3-1.

Listing 3-1 Serving HTTP Live Streaming in a webpage

```
<html>
<head>
    <title>HTTP Live Streaming Example</title>
</head>
<body>
    <video
        src="http://devimages.apple.com/iphone/samples/bipbop/bipbopall.m3u8"
        height="300" width="400"
    >
    </video>
</body>
```

```
</html>
```

For browsers that don't support the HTML5 `video` element, or browsers that don't support HTTP Live Streaming, you can include fallback code between the `<video>` and `</video>` tags. For example, you could fall back to a progressive download movie or an RTSP stream using the QuickTime plug-in. See *Safari HTML5 Audio and Video Guide* for examples.

Configuring a Web Server

HTTP Live Streaming can be served from an ordinary web server; no special configuration is necessary, apart from associating the MIME types of the files being served with their file extensions.

Configure the following MIME types for HTTP Live Streaming:

File Extension	MIME Type
`.M3U8`	`application/x-mpegURL` or `vnd.apple.mpegURL`
`.ts`	`video/MP2T`

If your web server is constrained with respect to MIME types, you can serve files ending in `.m3u` with MIME type `audio/mpegURL` for compatibility.

Index files can be long and may be frequently redownloaded, but they are text files and can be compressed very efficiently. You can reduce server overhead by enabling on-the-fly `.gzip` compression of `.M3U8` index files; the HTTP Live Streaming client automatically unzips compressed index files.

Shortening time-to-live (TTL) values for `.M3U8` files may also be needed to achieve proper caching behavior for downstream web caches, as these files are frequently overwritten during live broadcasts, and the latest version should be downloaded for each request. Check with your content delivery service provider for specific recommendations. For VOD, the index file is static and downloaded only once, so caching is not a factor.

Validating Your Streams

The `mediastreamvalidator` tool is a command-line utility for validating HTTP Live Streaming streams and servers (see "Download the Tools" (page 14) for details on obtaining the tool).

The media stream validator simulates an HTTP Live Streaming session and verifies that the index file and media segments conform to the HTTP Live Streaming specification. It performs several checks to ensure reliable streaming. If any errors or problems are found, a detailed diagnostic report is displayed.

You should always run the validator prior to serving a new stream or alternate stream set.

The media stream validator shows a listing of the streams you provide, followed by the timing results for each of those streams. (It may take a few minutes to calculate the actual timing.) An example of validator output follows.

```
$ mediastreamvalidator -d iphone
http://devimages.apple.com/iphone/samples/bipbop/gear3/prog_index.m3u8
mediastreamvalidator: Beta Version 1.1(130423)


Validating http://devimages.apple.com/iphone/samples/bipbop/gear3/prog_index.m3u8


________________________________________________________________________________
http://devimages.apple.com/iphone/samples/bipbop/gear3/prog_index.m3u8
________________________________________________________________________________


Playlist Syntax:     OK


Segments:    OK


Average segment duration: 9.91 seconds
Segment bitrate: Average: 509.56 kbits/sec, Max: 840.74 kbits/sec
Average segment structural overhead: 97.49 kbits/sec (19.13 %)
```

For more information, read *Media Stream Validator Tool Results Explained*.

Serving Key Files Securely Over HTTPS

You can protect your media by encrypting it. The file segmenter and stream segmenter both have encryption options, and you can tell them to change the encryption key periodically. Who you share the keys with is up to you.

Key files require an initialization vector (IV) to decode encrypted media. The IVs can be changed periodically, just as the keys can. Current recommendations for encrypting media while minimizing overhead is to change the key every 3-4 hours and change the IV after every 50 Mb of data.

Even with restricted access to keys, however, it is possible for an eavesdropper to obtain copies of the key files if they are sent over HTTP. One solution to this problem is to send the keys securely over HTTPS.

Before you attempt to serve key files over HTTPS, you should do a test serving the keys from an internal web server over HTTP. This allows you to debug your setup before adding HTTPS to the mix. Once you have a known working system, you are ready to make the switch to HTTPS.

There are three conditions you must meet in order to use HTTPS to serve keys for HTTP Live Streaming:

- You need to install an SSL certificate signed by a trusted authority on your HTTPS server.
- The authentication domain for the key files must be the same as the authentication domain for the first playlist file. The simplest way to accomplish this is to serve the variant playlist file from the HTTPS server—the variant playlist file is downloaded only once, so this shouldn't cause an excessive burden. Other playlist files can be served using HTTP.
- You must either initiate your own dialog for the user to authenticate, or you must store the credentials on the client device—HTTP Live Streaming does not provide user dialogs for authentication. If you are writing your own client app, you can store credentials, whether cookie-based or HTTP digest based, and supply the credentials in the `didReceiveAuthenticationChallenge` callback (see "Using NSURLConnection" and "Authentication Challenges" for details). The credentials you supply are cached and reused by the media player.

Important: You must obtain an SSL certificate signed by a trusted authority in order to use an HTTPS server with HTTP Live Streaming.

If your HTTPS server does not have an SSL certificate signed by a trusted authority, you can still test your setup by creating a self-signed SSL Certificate Authority and a leaf certificate for your server. Attach the certificate for the certificate authority to an email, send it to a device you want to use as a Live Streaming client, and tap on the attachment in Mail to make the device trust the server.

Frequently Asked Questions

1. **What kinds of encoders are supported?**

   The protocol specification does not limit the encoder selection. However, the current Apple implementation should interoperate with encoders that produce MPEG-2 Transport Streams containing H.264 video and AAC audio (HE-AAC or AAC-LC). Encoders that are capable of broadcasting the output stream over UDP should also be compatible with the current implementation of the Apple provided segmenter software.

2. **What are the specifics of the video and audio formats supported?**

   Although the protocol specification does not limit the video and audio formats, the current Apple implementation supports the following formats:

   - Video:
     - H.264 Baseline Level 3.0, Baseline Level 3.1, Main Level 3.1, and High Profile Level 4.1.
   - Audio:
     - HE-AAC or AAC-LC up to 48 kHz, stereo audio
     - MP3 (MPEG-1 Audio Layer 3) 8 kHz to 48 kHz, stereo audio
     - AC-3 (for Apple TV, in pass-through mode only)

   Note: iPad, iPhone 3G, and iPod touch (2nd generation and later) support H.264 Baseline 3.1. If your app runs on older versions of iPhone or iPod touch, however, you should use H.264 Baseline 3.0 for compatibility. If your content is intended solely for iPad, Apple TV, iPhone 4 and later, and Mac OS X computers, you should use Main Level 3.1.

3. **What duration should media files be?**

   The main point to consider is that shorter segments result in more frequent refreshes of the index file, which might create unnecessary network overhead for the client. Longer segments will extend the inherent latency of the broadcast and initial startup time. A duration of 10 seconds of media per file seems to strike a reasonable balance for most broadcast content.

4. **How many files should be listed in the index file during a continuous, ongoing session?**

   The normal recommendation is 3, but the optimum number may be larger.

The important point to consider when choosing the optimum number is that the number of files available during a live session constrains the client's behavior when doing play/pause and seeking operations. The more files in the list, the longer the client can be paused without losing its place in the broadcast, the further back in the broadcast a new client begins when joining the stream, and the wider the time range within which the client can seek. The trade-off is that a longer index file adds to network overhead—during live broadcasts, the clients are all refreshing the index file regularly, so it does add up, even though the index file is typically small.

5. **What data rates are supported?**

   The data rate that a content provider chooses for a stream is most influenced by the target client platform and the expected network topology. The streaming protocol itself places no limitations on the data rates that can be used. The current implementation has been tested using audio-video streams with data rates as low as 64 Kbps and as high as 3 Mbps to iPhone. Audio-only streams at 64 Kbps are recommended as alternates for delivery over slow cellular connections.

   For recommended data rates, see "Preparing Media for Delivery to iOS-Based Devices" (page 23).

   > **Note:** If the data rate exceeds the available bandwidth, there is more latency before startup and the client may have to pause to buffer more data periodically. If a broadcast uses an index file that provides a moving window into the content, the client will eventually fall behind in such cases, causing one or more segments to be dropped. In the case of VOD, no segments are lost, but inadequate bandwidth does cause slower startup and periodic stalling while data buffers.

6. **What is a .ts file?**

   A `.ts` file contains an MPEG-2 Transport Stream. This is a file format that encapsulates a series of encoded media samples—typically audio and video. The file format supports a variety of compression formats, including MP3 audio, AAC audio, H.264 video, and so on. Not all compression formats are currently supported in the Apple HTTP Live Streaming implementation, however. (For a list of currently supported formats, see "Media Encoder" (page 10).

   MPEG-2 Transport Streams are containers, and should not be confused with MPEG-2 compression.

7. **What is an .M3U8 file?**

   An `.M3U8` file is a extensible playlist file format. It is an m3u playlist containing UTF-8 encoded text. The m3u file format is a de facto standard playlist format suitable for carrying lists of media file URLs. This is the format used as the index file for HTTP Live Streaming. For details, see IETF Internet-Draft of the HTTP Live Streaming specification.

8. **How does the client software determine when to switch streams?**

The current implementation of the client observes the effective bandwidth while playing a stream. If a higher-quality stream is available and the bandwidth appears sufficient to support it, the client switches to a higher quality. If a lower-quality stream is available and the current bandwidth appears insufficient to support the current stream, the client switches to a lower quality.

> **Note:** For seamless transitions between alternate streams, the audio portion of the stream should be identical in all versions.

9. **Where can I find a copy of the media stream segmenter from Apple?**

   The media stream segmenter, file stream segmenter, and other tools are frequently updated, so you should download the current version of the HTTP Live Streaming Tools from the Apple Developer website. See "Download the Tools" (page 14) for details.

10. **What settings are recommended for a typical HTTP stream, with alternates, for use with the media segmenter from Apple?**

    See "Preparing Media for Delivery to iOS-Based Devices" (page 23).

    These settings are the current recommendations. There are also certain requirements. The current `mediastreamsegmenter` tool works only with MPEG-2 Transport Streams as defined in ISO/IEC 13818. The transport stream must contain H.264 (MPEG-4, part 10) video and AAC or MPEG audio. If AAC audio is used, it must have ADTS headers. H.264 video access units must use Access Unit Delimiter NALs, and must be in unique PES packets.

    The segmenter also has a number of user-configurable settings. You can obtain a list of the command line arguments and their meanings by typing `man mediastreamsegmenter` from the Terminal application. A target duration (length of the media segments) of 10 seconds is recommended, and is the default if no target duration is specified.

11. **How can I specify what codecs or H.264 profile are required to play back my stream?**

    Use the `CODECS` attribute of the `EXT-X-STREAM-INF` tag. When this attribute is present, it must include all codecs and profiles required to play back the stream. The following values are currently recognized:

AAC-LC	`"mp4a.40.2"`
HE-AAC	`"mp4a.40.5"`
MP3	`"mp4a.40.34"`
H.264 Baseline Profile level 3.0	`"avc1.42001e"` or `"avc1.66.30"` Note: Use `"avc1.66.30"` for compatibility with iOS versions 3.0 to 3.1.2.

H.264 Baseline Profile level 3.1	`"avc1.42001f"`
H.264 Main Profile level 3.0	`"avc1.4d001e"` or `"avc1.77.30"` Note: Use `"avc1.77.30"` for compatibility with iOS versions 3.0 to 3.12.
H.264 Main Profile level 3.1	`"avc1.4d001f"`
H.264 Main Profile level 4.0	`"avc1.4d0028"`
H.264 High Profile level 3.1	`"avc1.64001f"`
H.264 High Profile level 4.0	`"avc1.640028"`
H.264 High Profile level 4.0	`"avc1.640029"`

The attribute value must be in quotes. If multiple values are specified, one set of quotes is used to contain all values, and the values are separated by commas. An example follows.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000, RESOLUTION=720x480
mid_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=800000, RESOLUTION=1280x720
wifi_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=3000000,
CODECS="avc1.4d001e,mp4a.40.5", RESOLUTION=1920x1080
h264main_heaac_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=64000, CODECS="mp4a.40.5"
aacaudio_index.M3U8
```

12. **How can I create an audio-only stream from audio/video input?**

    Add the `-audio-only` argument when invoking the stream or files segmenter.

13. **How can I add a still image to an audio-only stream?**

    Use the `-meta-file` argument when invoking the stream or file segmenter with `-meta-type=picture` to add an image to every segment. For example, this would add an image named poster.jpg to every segment of an audio stream created from the file track01.mp3:

    `mediafilesegmenter -f` *`/Dir/outputFile`* `-a --meta-file=poster.jpg --meta-type=picture track01.mp3`

    Remember that the image is typically resent every ten seconds, so it's best to keep the file size small.

14. **How can I specify an audio-only alternate to an audio-video stream?**

Use the `CODECS` and `BANDWIDTH` attributes of the `EXT-X-STREAM-INF` tag together.

The `BANDWIDTH` attribute specifies the bandwidth required for each alternate stream. If the available bandwidth is enough for the audio alternate, but not enough for the lowest video alternate, the client switches to the audio stream.

If the `CODECS` attribute is included, it must list all codecs required to play the stream. If only an audio codec is specified, the stream is identified as audio-only. Currently, it is not required to specify that a stream is audio-only, so use of the `CODECS` attribute is optional.

The following is an example that specifies video streams at 500 Kbps for fast connections, 150 Kbps for slower connections, and an audio-only stream at 64 Kbps for very slow connections. All the streams should use the same 64 Kbps audio to allow transitions between streams without an audible disturbance.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000, RESOLUTION=1920x1080
mid_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=150000, RESOLUTION=720x480
3g_video_index.M3U8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=64000, CODECS="mp4a.40.5"
aacaudio_index.M3U8
```

15. **What are the hardware requirements or recommendations for servers?**

    See question #1 for encoder hardware recommendations.

    The Apple stream segmenter is capable of running on any Intel-based Mac. We recommend using a Mac with two Ethernet network interfaces, such as a Mac Pro or an XServe. One network interface can be used to obtain the encoded stream from the local network, while the second network interface can provide access to a wider network.

16. **Does the Apple implementation of HTTP Live Streaming support DRM?**

    No. However, media can be encrypted, and key access can be limited by requiring authentication when the client retrieves the key from your HTTPS server.

17. **What client platforms are supported?**

    iPhone, iPad, and iPod touch (requires iOS version 3.0 or later), Apple TV (version 2 and later), and Mac OS X computers.

18. **Is the protocol specification available?**

    Yes. The protocol specification is an IETF Internet-Draft, at http://tools.ietf.org/html/draft-pantos-http-live-streaming.

19. **Does the client cache content?**

The index file can contain an instruction to the client that content should not be cached. Otherwise, the client may cache data for performance optimization when seeking within the media.

20. **Is this a real-time delivery system?**

    No. It has inherent latency corresponding to the size and duration of the media files containing stream segments. At least one segment must fully download before it can be viewed by the client, and two may be required to ensure seamless transitions between segments. In addition, the encoder and segmenter must create a file from the input; the duration of this file is the minimum latency before media is available for download. Typical latency with recommended settings is in the neighborhood of 30 seconds.

21. **What is the latency?**

    Approximately 30 seconds, with recommended settings. See question #15.

22. **Do I need to use a hardware encoder?**

    No. Using the protocol specification, it is possible to implement a software encoder.

23. **What advantages does this approach have over RTP/RTSP?**

    HTTP is less likely to be disallowed by routers, NAT, or firewall settings. No ports need to be opened that are commonly closed by default. Content is therefore more likely to get through to the client in more locations and without special settings. HTTP is also supported by more content-distribution networks, which can affect cost in large distribution models. In general, more available hardware and software works unmodified and as intended with HTTP than with RTP/RTSP. Expertise in customizing HTTP content delivery using tools such as PHP is also more widespread.

    Also, HTTP Live Streaming is supported in Safari and the media player framework on iOS. RTSP streaming is not supported.

24. **Why is my stream's overall bit rate higher than the sum of the audio and video bitrates?**

    MPEG-2 transport streams can include substantial overhead. They utilize fixed packet sizes that are padded when the packet contents are smaller than the default packet size. Encoder and multiplexer implementations vary in their efficiency at packing media data into these fixed packet sizes. The amount of padding can vary with frame rate, sample rate, and resolution.

25. **How can I reduce the overhead and bring the bit rate down?**

    Using a more efficient encoder can reduce the amount of overhead, as can tuning the encoder settings.

26. **Do all media files have to be part of the same MPEG-2 Transport Stream?**

    No. You can mix media files from different transport streams, as long as they are separated by `EXT-X-DISCONTINUITY` tags. See the protocol specification for more detail. For best results, however, all video media files should have the same height and width dimensions in pixels.

27. **Where can I get help or advice on setting up an HTTP audio/video server?**

    You can visit the Apple Developer Forum at http://devforums.apple.com/.

Also, check out *Best Practices for Creating and Deploying HTTP Live Streaming Media for the iPhone and iPad* .

Document Revision History

This table describes the changes to *HTTP Live Streaming Overview*.

Date	Notes
2013-08-08	Made minor technical corrections.
2011-04-01	Added descripton of closed caption support, best practices for preparing media for iOS-based devices, and description of how to serve key files over HTTPS.
2010-11-15	Added description of timed metadata.
2010-03-25	Updated to include iPad and -optimize option for stream segmenter.
2010-02-05	Defines requirement for apps to use HTTP Live Streaming for video over cellular networks and requirement for 64 Kbps streams.
2010-01-20	Updated links to tools and sample streams. Added documentation of CODECS attribute and video streams with audio-only alternates.
2009-11-17	Fixed typos and corrected URLs for samples.
2009-09-09	Updated to include failover support.
2009-06-04	Added sample code, QuickTime X support. Reorganized document for readability.
2009-05-22	Changed title from iPhone Streaming Media Guide for Web Developers. Added URL for downloading media segmenter.
2009-03-15	New document that describes live streaming of audio and video over HTTP for iPhone.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010