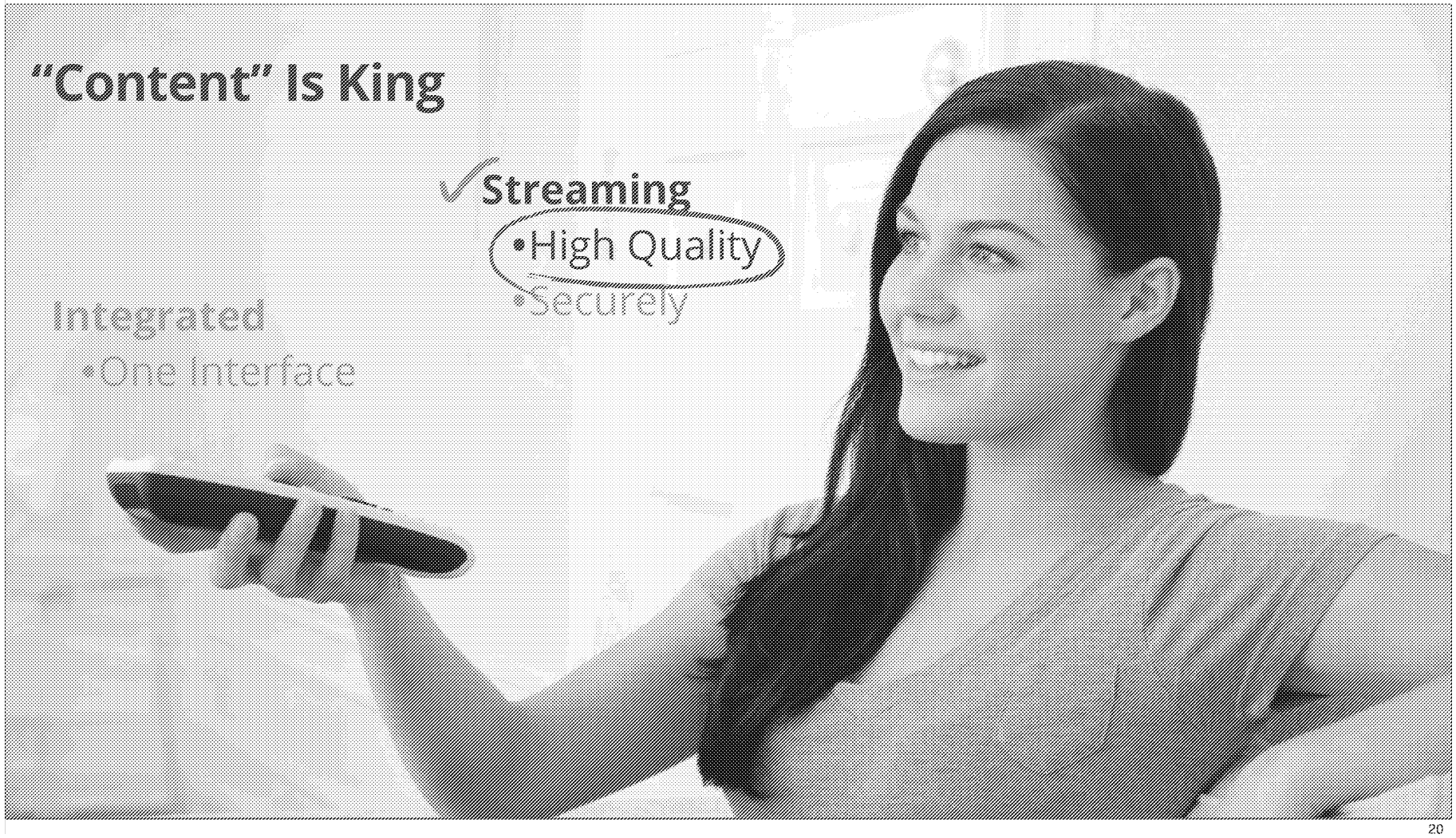

"Content" Is King
✓Streaming
•High Quality
•Securely
Integrated
•One Interface
20

Ensuring high quality delivery

For best results, you want to continuously *monitor*:

- Network bandwidth
- Playback *quality* (Frames per second, FPS and dropped frames)

This allows you to:

- Ensure **customer satisfaction**
  - Offer refunds or re-play
- Proactively detect (and mitigate) **connectivity problems**
  - "Sorry, your Internet connection does not support 1080p playback!"



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0765
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

We Introduce Quality of Service (QoS) APIs

Now you can measure:

- Frames per Second (**FPS**)
- Network **Bandwidth**
- Buffer size, **Buffer fill rate**
- Buffered media playback duration
- Audio Info
- Detailed errors from underlying system, Unexpected End Of Stream (EOS)

Use "**Analytics**" frameworks to aggregate Playback Quality statistics



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0766
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Example using QoS API

- Implement a **MediaPlayer.OnInfoListener** class
- Register with **GtvVideoView.setOnInfoListener**( ).

```
private class InfoListener implements MediaPlayer.OnInfoListener {

   public boolean onInfo(MediaPlayer mediaPlayer, int what, int extra) {

   switch (what) {
       case GtvMediaPlayer.MEDIA_INFO_NETWORK_BANDWIDTH:
          Log.d(TAG, "Current Bandwidth: " + extra + " kbps"); break;

       case GtvMediaPlayer.MEDIA_INFO_FPS: {
          Log.d(TAG, "Current FPS: " + extra);
                                                  // In real world, check instanceof()
          MediaPlayer gtvPlayer = (GtvMediaPlayer) mediaPlayer;
                                                  // get current media playback info
          OnInfoMetaData mediaInfo= gtvPlayer.getOnInfoMetaData();
          int droppedFPS = mediaInfo.getInteger(OnInfoMetadata.MEDIA_INFO_META_DROPPED_FPS);

          Log.d(TAG, "Dropped FPS: " + droppedFPS);
       }
}
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0767
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Developers first!

We will provide:

- An implementation of **Smooth Streaming protocol** with
  - Media Source API
  - PlayReady DRM handling in Java
- Multi-track audio handling
- TTML based closed caption handling
- Demonstration of QoS APIs to monitor streaming quality

We will release **sample code** how to use all those APIs as **Open Source**.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0768
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

But Wait... There is More!

Google TV apps can now play back **High Quality Content** directly from **YouTube**

If you want to utilize **OUR** existing **encoding** and **streaming servers** inside **YOUR** app:

Use the **YouTube Android Player API** for:

- Playing videos and playlists
- Registering to be notified of playback events
- YouTube UI widgets (e.g., YouTubePlayerView, VideoThumbnailView)

Session: **YouTube Player API**, Thursday at 1:30 PM



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0769
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0770
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

DRM - In a Nutshell

Digital Rights Management:

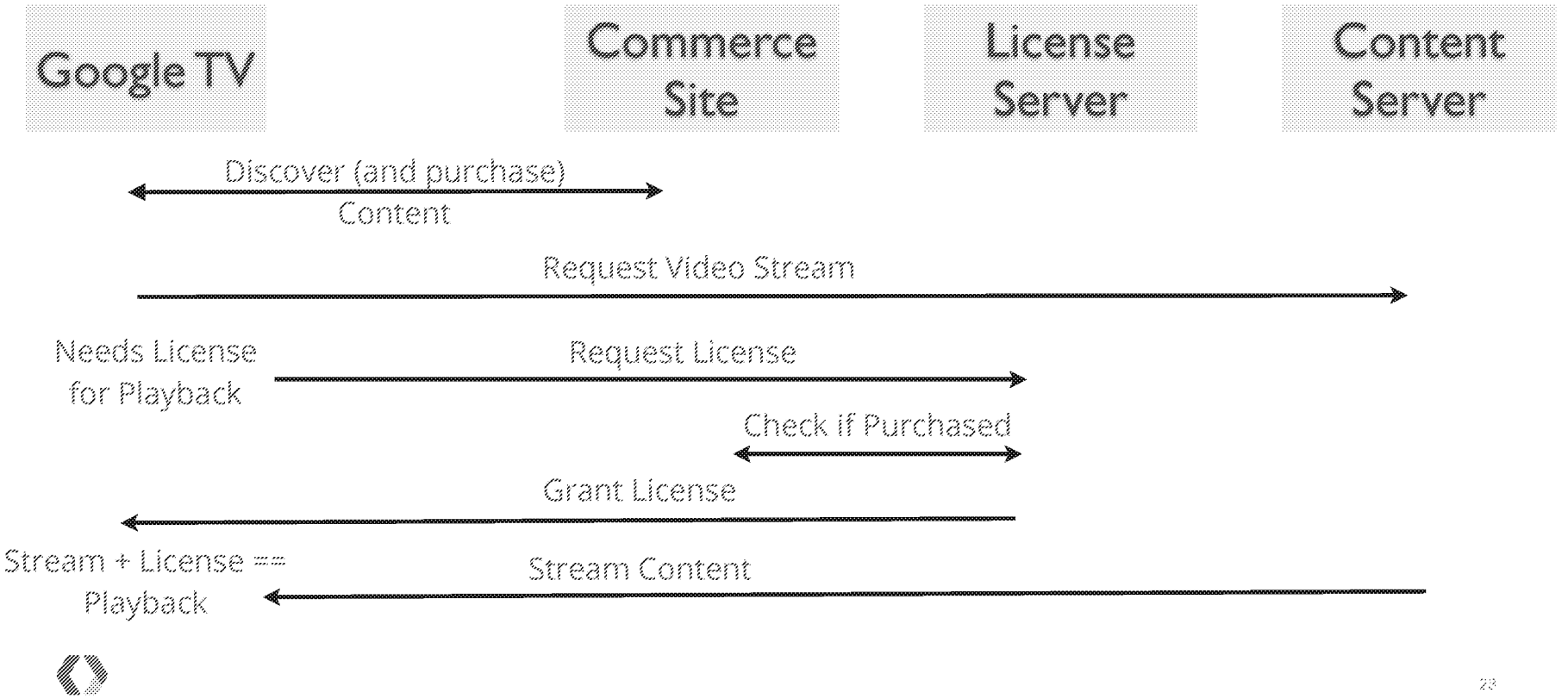
- License Management
- Transport Encryption
- Secure Decoding (Trusted Video Path)

***HTTPS* is not a DRM**



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0771
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Typical DRM Use Case



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0772
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

DRM - In Android

DRM Framework introduced in Android 3.0 (HoneyComb)

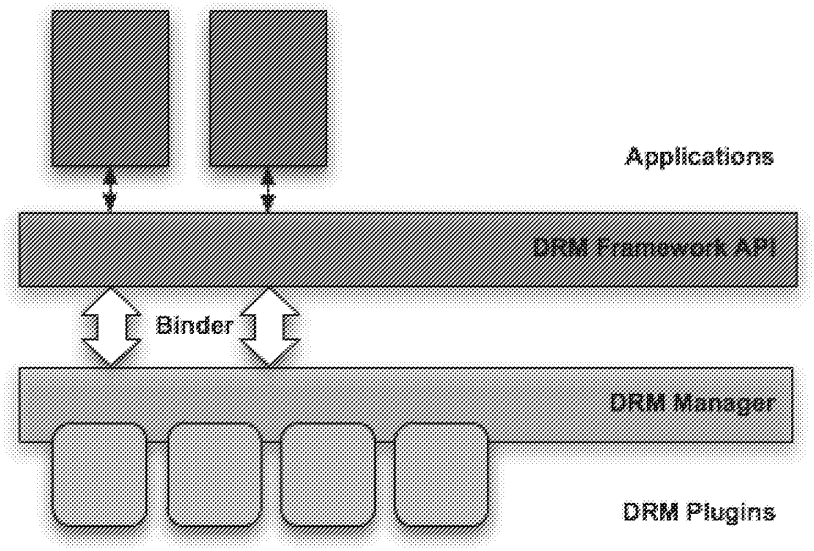
- Extensible
- Android Application interact with a "Native Code" DRM Provider implementation

Android DRM Framework in package: android.drm

Android has built-in support for Widevine DRM.

But: Custom DRM Plugins

- Require Native Code
- Difficult to develop





Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0773
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Introducing PlayReady DRM for Google TV

ARM based Google TV devices will include support for **Microsoft PlayReady DRM**:

- Implemented as a **plug-in** to Android DRM Framework
- Accessed via standard **Android DRM** (Java) **APIs**
- Supports basic **license acquisition** and **management**
  - **Extensible** to adapt to custom license servers and protocols.
- Playback using hardware "**Trusted Video Path**" (TVP)

- Integrated with Smooth Streaming protocol



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0774
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Google TV - Keeping Your Content Safe

Google TV DRM Components:

- Android **DRM Framework**
  - Managing playback rights of content
  - **Widevine** and **PlayReady** support now built into the platform

- **Trusted Video Path** (TVP)
  - Keeps decrypted video data securely in hardware secure "sandbox"
  - Protects streamed media securely

- **HDMI Content Protection**
  - Protecting Video Content all the way to the Television display



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0775
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0776
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The "User Interface" To Your Entertainment
Internet
PC

How Viewers Discover Content

Users **Search** for content in System Search

Users **Browse** for content in the TV & Movies app



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0778
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

How Google TV Finds Content

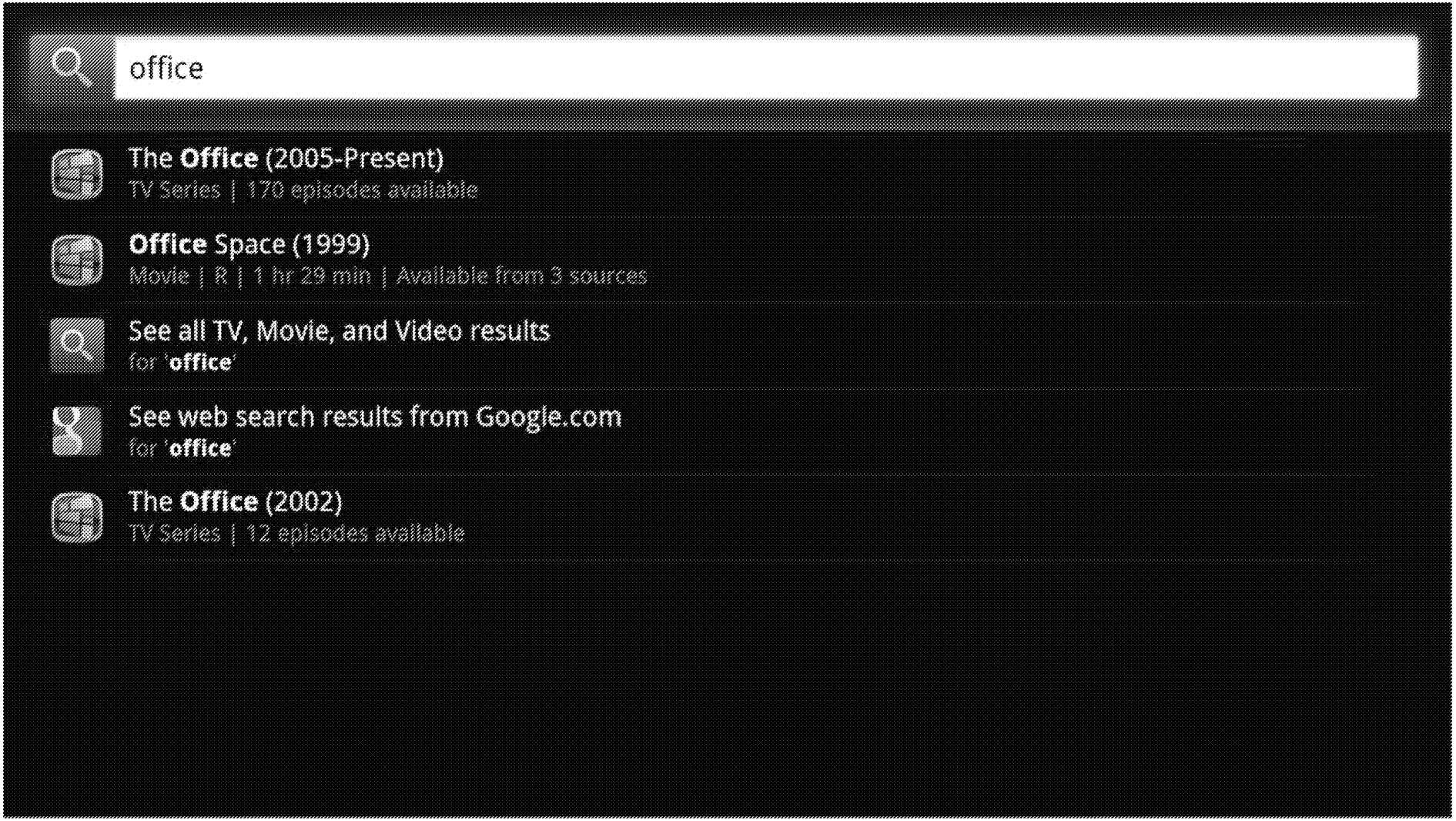
Content available on Google TV is an aggregate of:

- Backend:
  - Search engine indexed data: Video Site Map XML files
- Client side:
  - Media Devices (physical)
  - Media Devices (**virtual**)



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0779
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

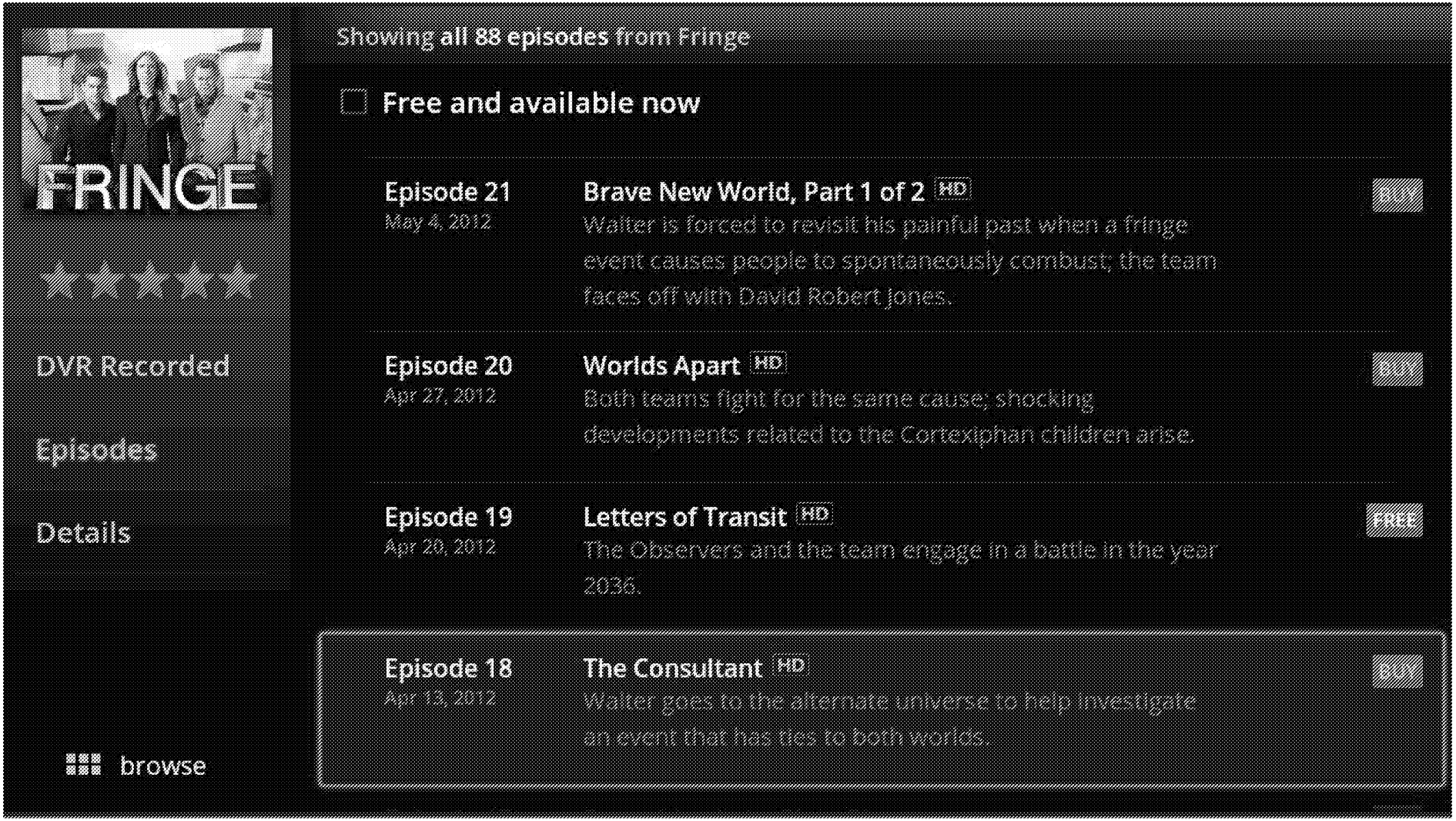
Media Device Interactions
Google TV
Channel Change
Channel List
DVR Recordings



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0781
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

12:41 PM
Favorite Channels
CNN NEWSROOM
CLOSING BELL
Countdown to the Closing Bell
Aired May 11, 2006. Stock market update.
Movies on TV
Comedy

38



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0783
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

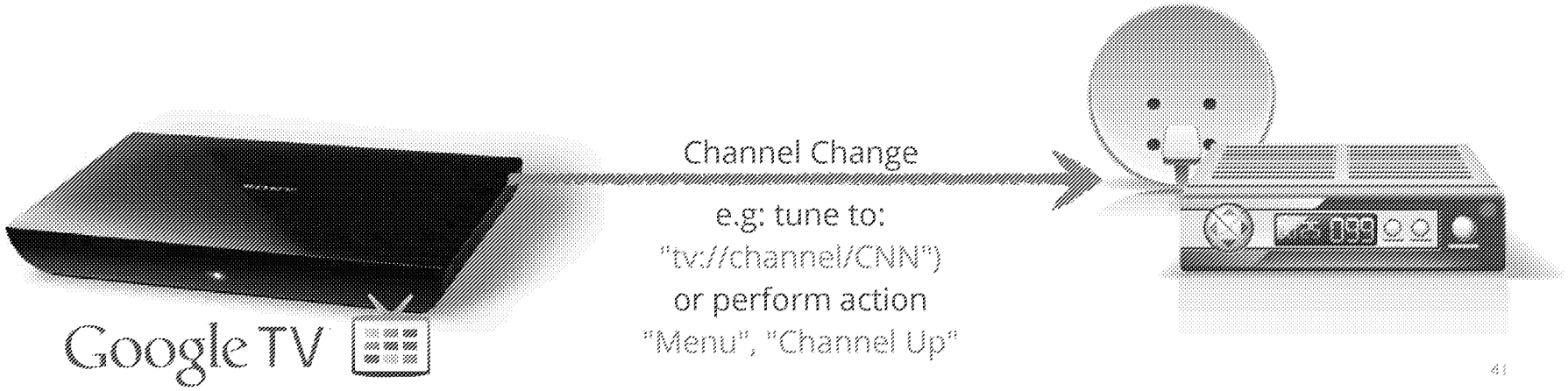
Your "device" HERE
Letters of Transit
Watch instantly
View details
View details
Rent now
HD
40

One Way Pairing

A (legacy) device that is controlled via an **Infra-Red** (IR) Blaster.

- When tighter integration is not available

Device cannot send information to GoogleTV.



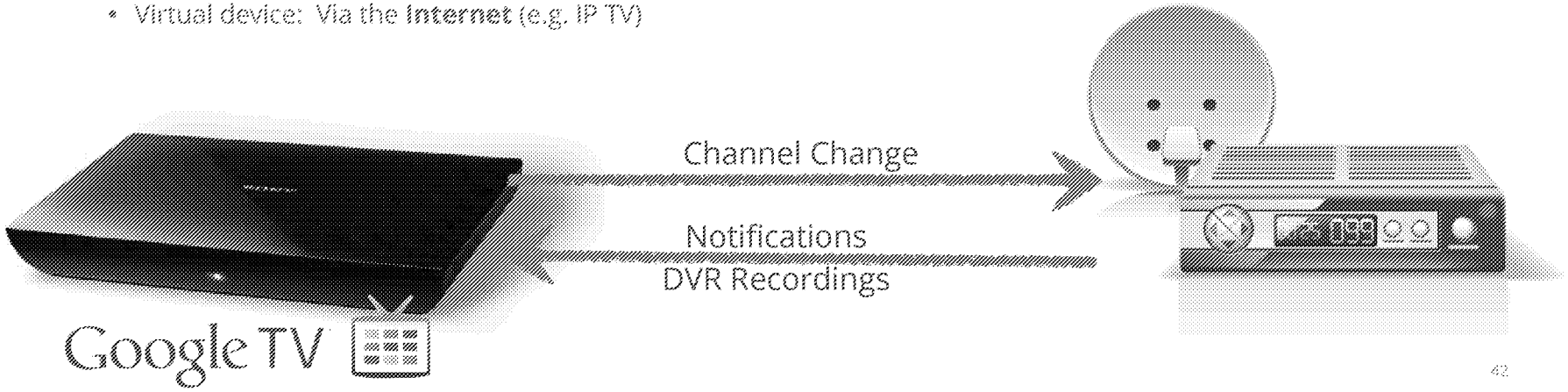
41

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0785
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Two Way Pairing

When a published communication protocol exists for controlling the device.

- Bidirectional communication
- Device can **send information** and **events** to GoogleTV
  - Channel change events
  - Closed caption text
- Frequently uses ***TCP/IP***
  - Physical device: **Locally** connected
  - Virtual device: Via the **Internet** (e.g. IP TV)

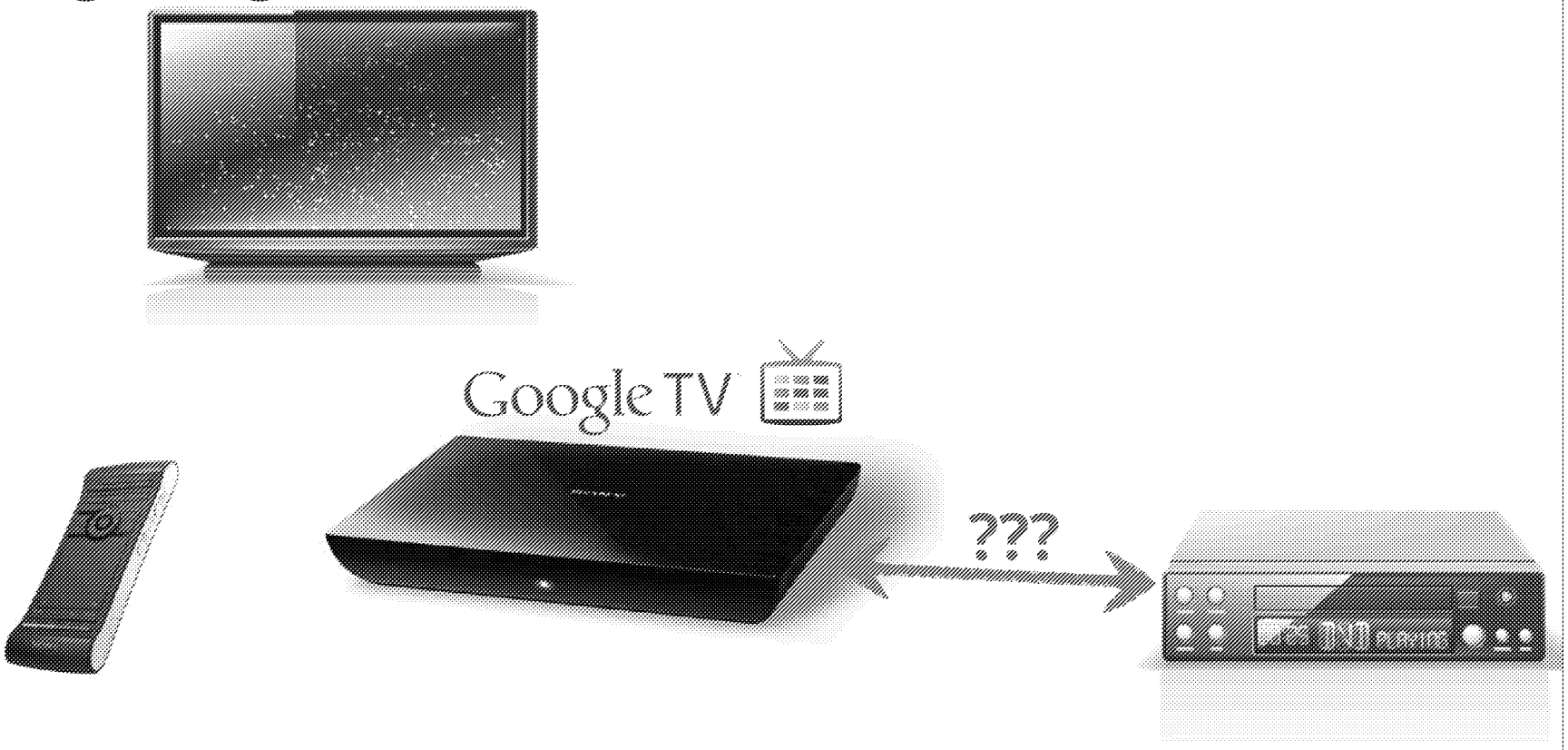


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0786
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0787
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Integrating New Media Devices



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0788
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Building Your Own Media Device

A Media Device is:

- Similar to a "Device **Driver**"
- **Packaged** as an **.apk**
- **Installed** from the **Google Play** store

Media Devices consist of the following software components:

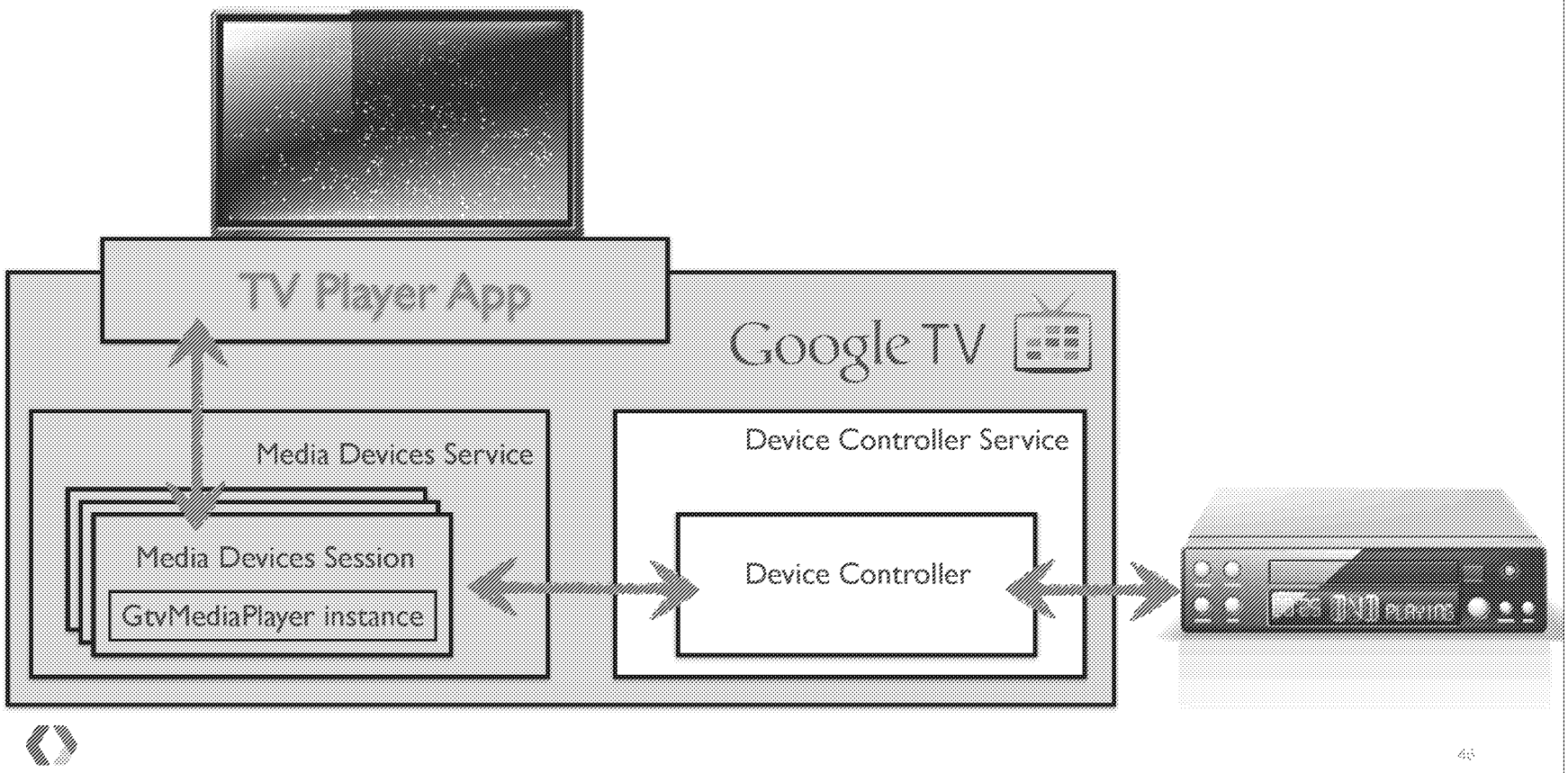
- **Media Device Controller Service** (Android Service)
- **Setup Activity**
- **Settings Activity**





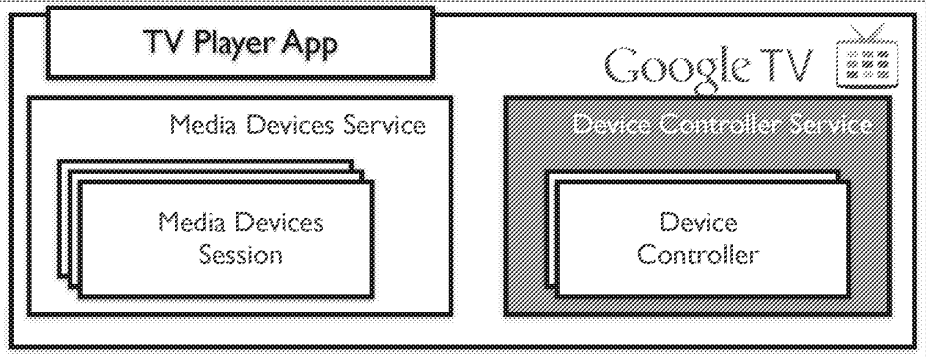
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0789
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Media Devices Framework Overview



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0790
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Device Controller Service



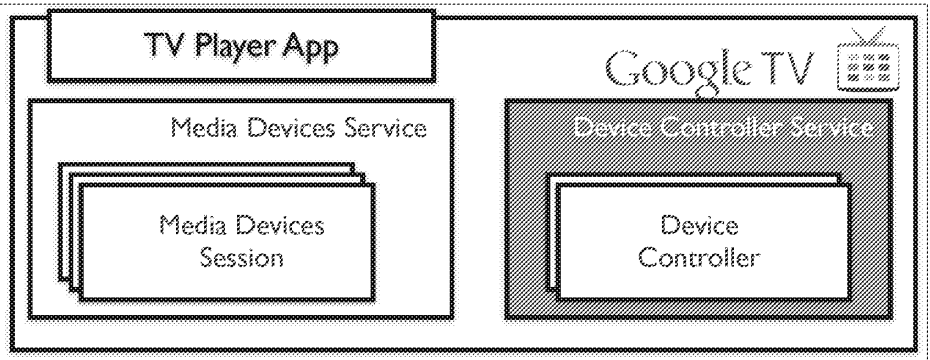
- Implemented as **Android service** (runs in the background)
- Interfaces to one or more media devices

- **Notifies the system:**
  - When devices go online / offline
  - Events like channel changes etc.
- **Reports** the devices **information:**
  - Channel lineups
  - Channel numbers / Call Signs
  - Names, logo icons



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0791
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Implementing a Device Controller Service
(Step 1/2)



Subclass **AbstractDeviceControllerService**

- Register the device(s) and their channel lineup(s) with the system:

Android

```
public final class MyDeviceService extends AbstractDeviceControllerService {
  public void onCreate() {
    super.onCreate();
    Device myDevice = buildDevice();
    setChannelUpdateInterval(myDevice.getId(), CHANNEL_UPDATE_INTERVAL_MS);
    addDevice(myDevice);
  }

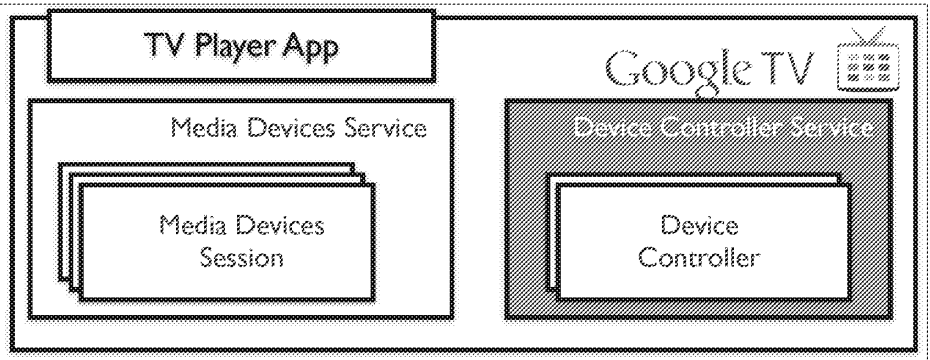
  protected final AbstractDeviceController buildController(final Device device) {
        return new MyDeviceController(this, getSettings(), device.getId());
  }
```



48

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0792
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Implementing a Device Controller Service
(Step 2/2)



Subclass **AbstractDeviceControllerService**

- Define and implement device features

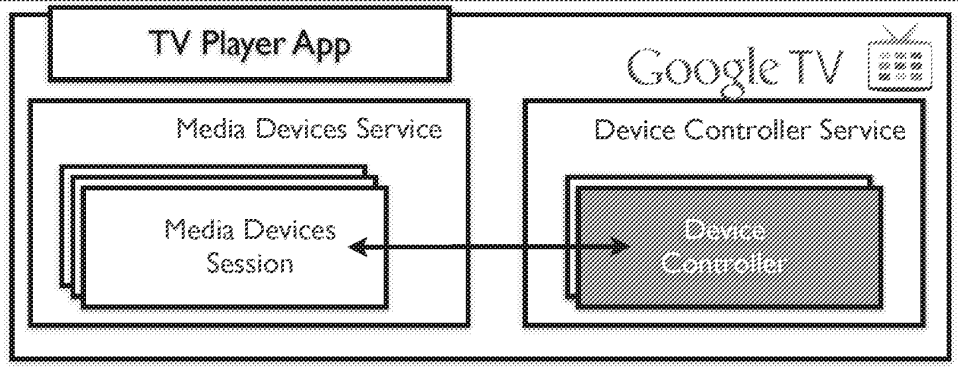
Android

```
private Device buildDevice() {
  String deviceId = getSettings().generateUniqueDeviceId("mydevice");
  return new Device.Builder(getPackageName(), deviceId)
    .setLabel(getString(R.string.my_device_label))
    .setCapability(Capability.CAN_DISCONNECT, true)
    .setCapability(Capability.HAS_CHANNEL_LINEUP, true)
    .setCapability(Capability.LOCK_CHANNEL_LINEUP, true)
    .build();
}
protected void checkForChannelUpdates(final String deviceId) {
    ...
}
}
```



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0793
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Device Controller



- Communicates to **device** using its **protocol**
- Handles user keypress **events**,
  - Channel Up, Fast Forward, Guide

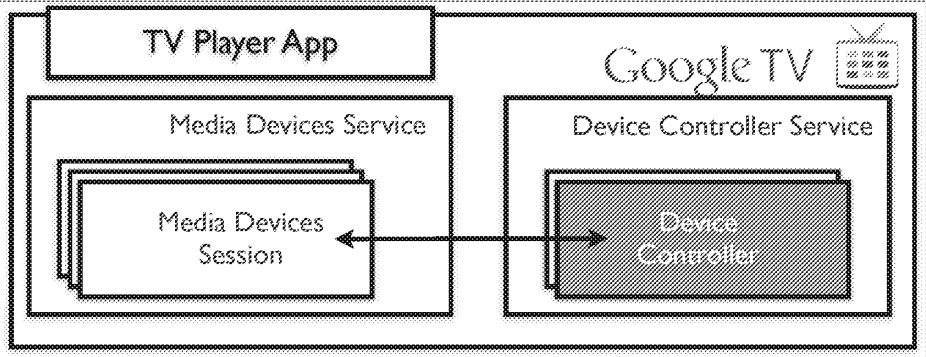
- Tunes device to requested **URIs** ("**tv://...**") which represent a channel, a VOD program or a DVR recording, etc.

- Controls the Media Player in the associated Device Session including start, stop playback and can change the players URI
  - e.g., "**hdmi://...**" URIs for passthrough devices
  - "**http://...**" URIs for "virtual" devices streaming video over the Internet



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0794
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Implementing a Device Controller



Subclass **AbstractDeviceController**

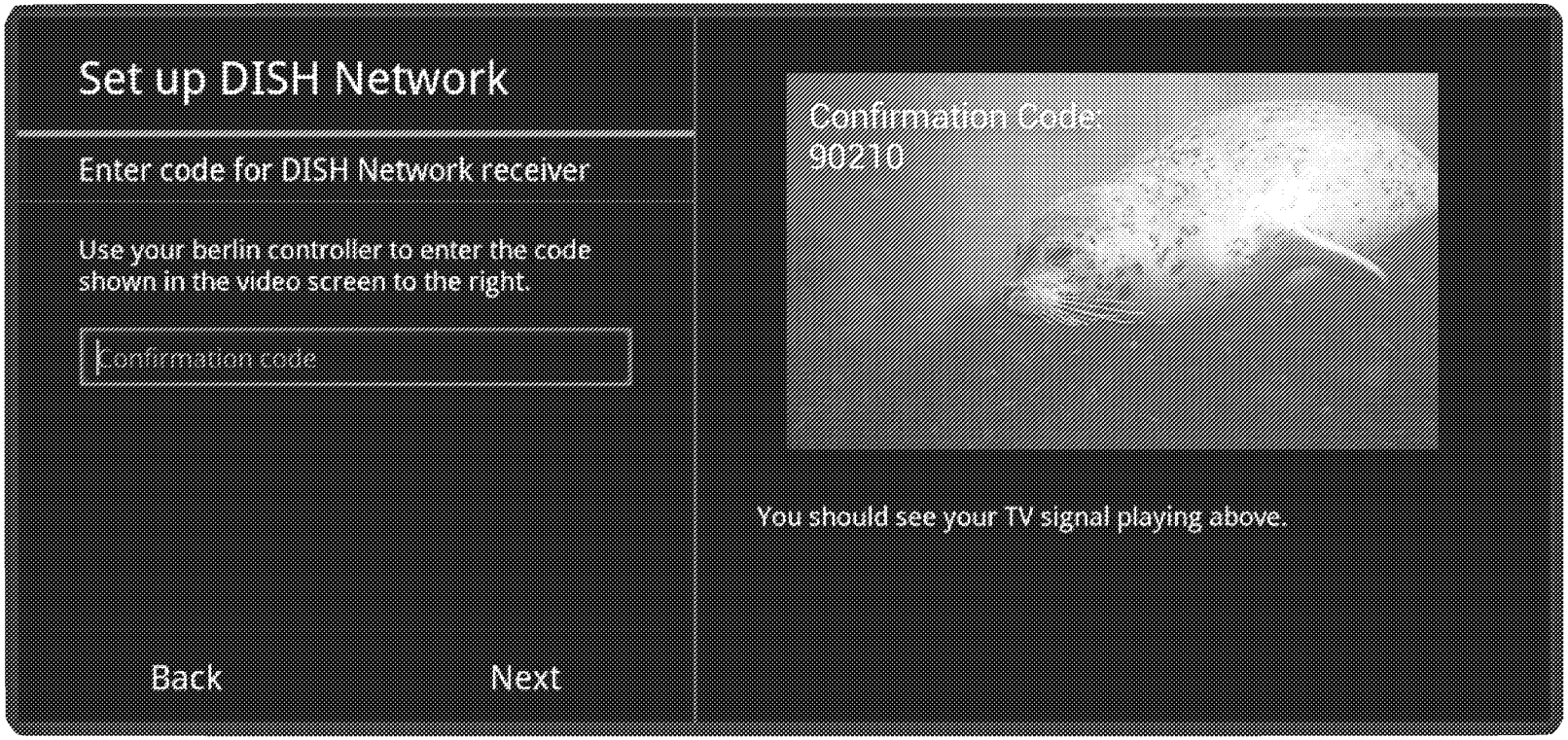
Android

```
public final class MyDeviceController extends AbstractDeviceController {
...
  public void performAction(ActionEvent event) {
    switch (event.getAction()) {
      case CHANNEL_UP:
        myNextChannel(); break;              // Implement necessary "channel up" change
      ...
    }
  }
  public void tuneToChannel(ChannelNumber channel) {
    Uri videoUri = ...;                      // Determine Video URI for the requested channel
    notifyLocationChanged(videoUri, EventSource.USER);   // inform the Session Media Player
  }
}
```



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0795
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Setup Activity / Pairing



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0796
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Settings Activity



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0797
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0798
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Bringing Content To Google TV:

- Supports many different Streaming Protocols
  - HTTP Live Streaming
  - Smooth Streaming
  - Optionally - allows to implement your own

- Compatible with Industry **Standard DRM** Solutions
  - Widevine
  - Smooth Streaming and PlayReady
  - Trusted Video Path

- Integrates with devices in your living room (and in the cloud)
  - Media Devices API



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0799
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Learn More About Google TV

http://developers.google.com/tv/

+Google TV Developers



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0800
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

<Thankyou/>

Google Developer Relations

+Christian Kurzke
+Andrew Nolan
+Mark Lindner

Google
Developers



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0803
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

H.264 or VC-1?

Customers should choose whichever codec best fits their encoding workflow and meets their quality requirements. However, keep in mind:

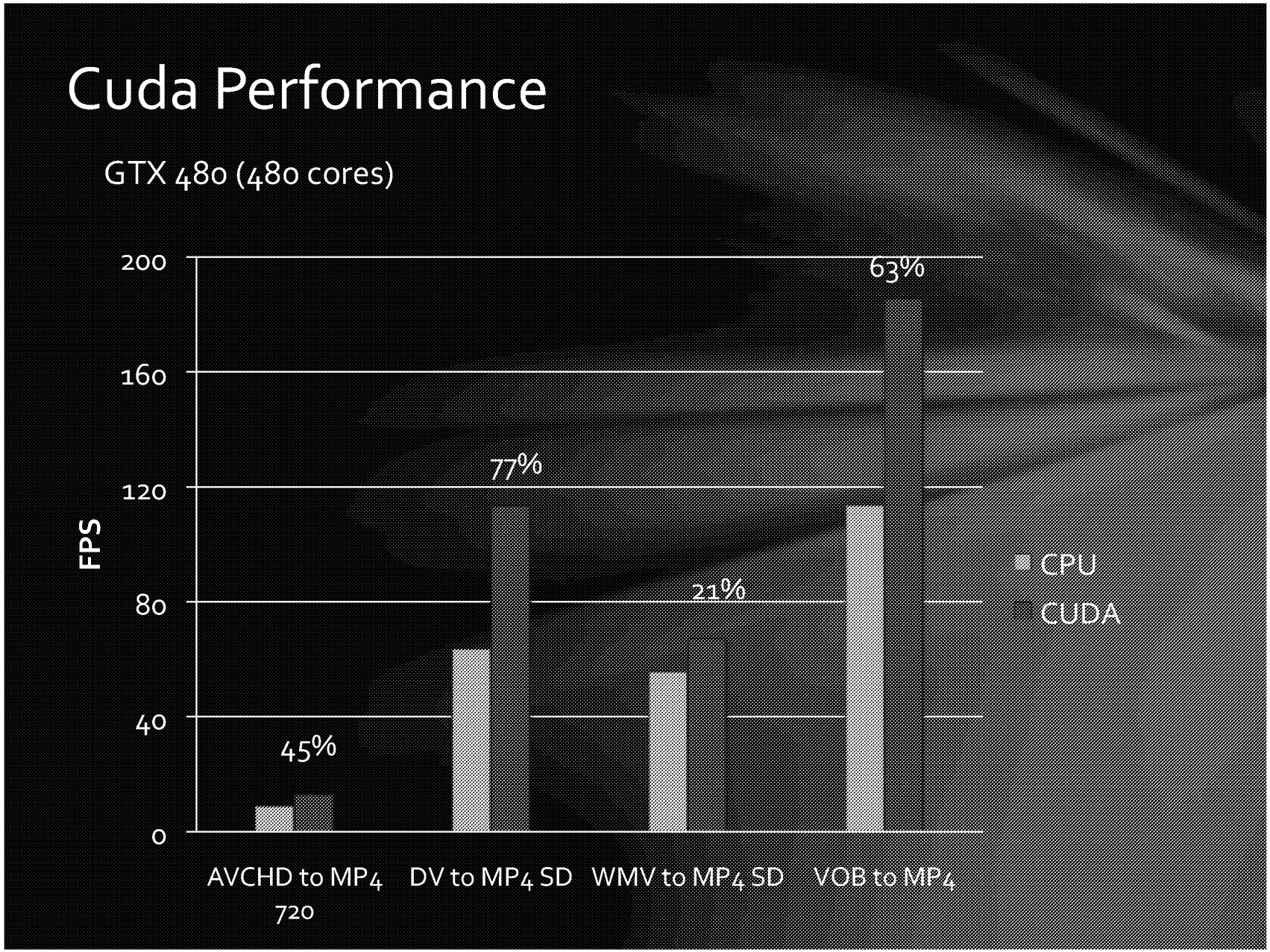
- H.264 decoding is typically more CPU intensive than VC-1 decoding for the same resolution and frame rate
- In Silverlight 4: H.264 decoding requires about 15-25% more CPU time than VC-1 decoding with similar content properties

Good rule of thumb:

- For HD video, use VC-1 to reach largest audience
- For SD video and smaller, use H.264 if quality gains are noticeable

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0804
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

H.264 Performance
FPS
30
25
20
15
10
5
0
V3
V4
57%
171%
176%
143%
AVCHD to MP4
720
DV to MP4 SD
WMV to MP4 SD
VOB to MP4



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0806
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Multi Bit Rate Encoding

- Smooth Streaming heavily utilizes the concept of encoding video at multiple bit rates and letting the client pick the most appropriate one
- Key differences between how MBR was used in the past and the way it's used in Smooth Streaming:
  - Resolution changes with bit rate in order to maintain consistent quality
  - Strict enforcement of video bitstream properties necessary to decode every chunk independently of other chunks
  - Temporal alignment of frames visually matches up with the same chunk in a different bit rate

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0807
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Why Change Resolution?

- Encoding the same resolution at inappropriately low bit rates introduces objectionable compression side effects into the video: blockiness, twirling details, color smearing, etc.
- By lowering the resolution proportionally to the bit rate we maintain a consistent level of compression quality in exchange for giving up some visual detail
- It's a compromise: between two evils, customers prefer a blurry picture over a blocky picture

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0808
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

VC-1: 640x360 at 1250 kbps
640x360 1250k

VC-1: 640x360 at 300 kbps
640x360 300k

VC-1: 288x160 at 300 kbps
288x160 300k

VC-1: 640x360 at 750 kbps
640x360 750k

VC-1: 400x224 at 750 kbps
400x224 750k

Determining Bit Rate Levels

- You can use as few as you'd like – even just a single level
- Upper bound is typically set by the encoding software
- 4 bit rates for SD and 6 bit rates for 720p HD are good starting points

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0814
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

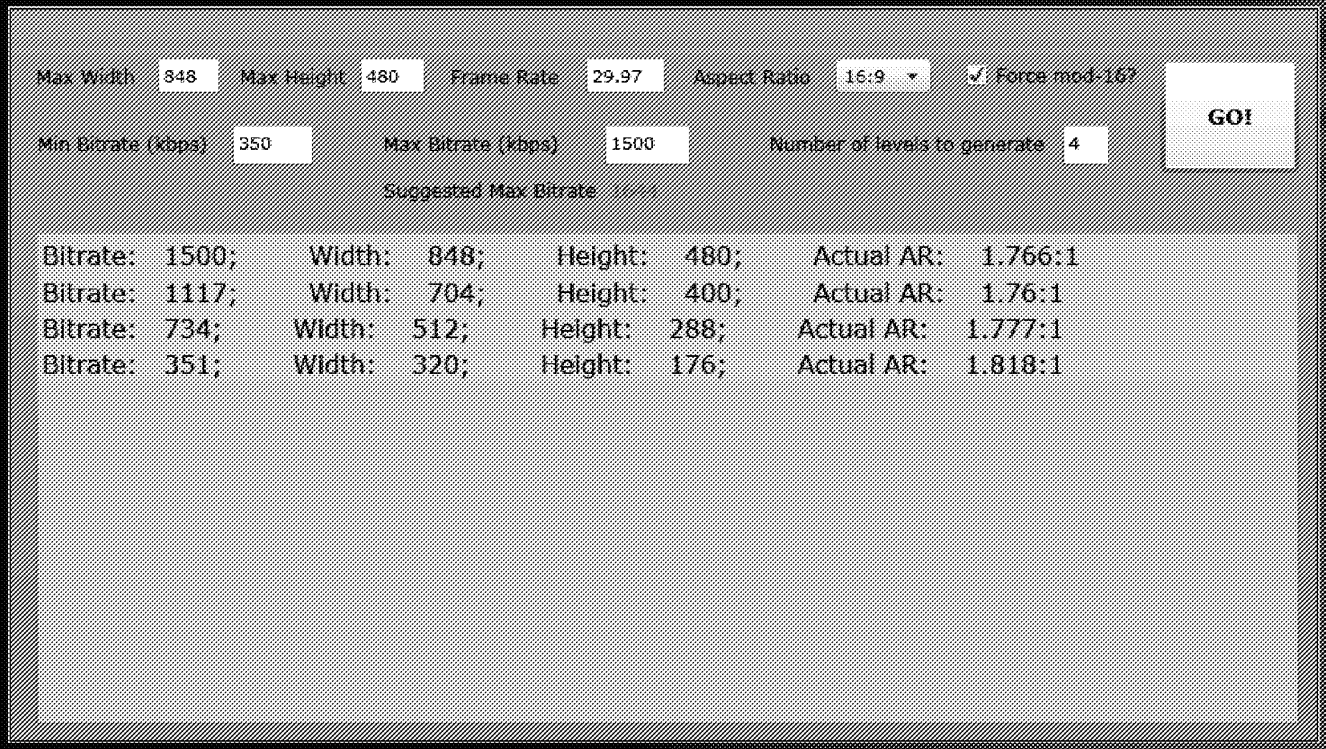
Picking Bit Rates

- Start by figuring out your maximum bit rate and minimum bit rate
- Allow about 250-500 kbps between bit rate levels
- Don't set minimum bit rate lower than 200 kbps
- Consider grouping the lower bitrates a little closer together than the rest because client bandwidth is most diverse on the lower end.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0815
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Smooth Streaming Calculator

http://alexzambelli.com/wmv/MBRCalc.html



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0816
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Encoding for Smooth Streaming

- Use VC-1 Advanced Profile or H.264 Main/High Profile
- 2 second key frame distance works very well
- Client heuristics are currently tuned for CBR content, so if using VBR set the peak bit rate within 15% of average bit rate for best playback
- Set video buffer size to 2x-3x key frame distance

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0817
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Demo

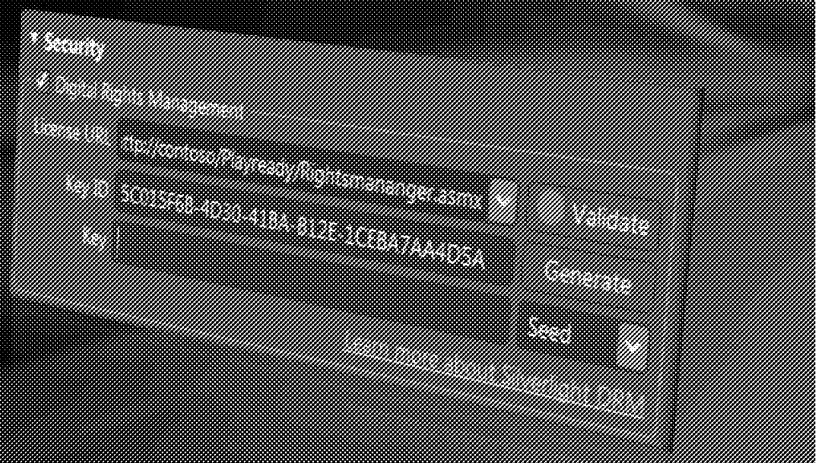
Live Smooth Streaming Performance Tool -

http://gallery.expression.microsoft.com/en-us/EE4LivePerfTool

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0818
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Digital Rights Management

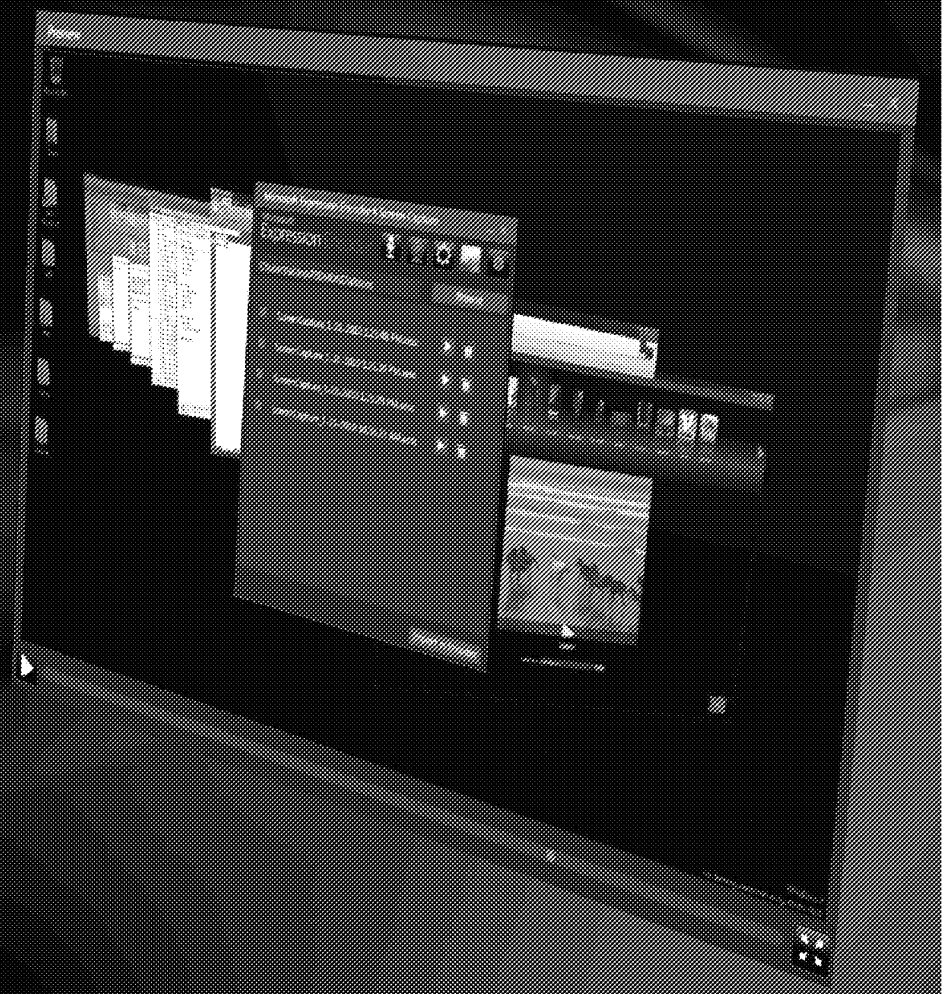
- PlayReady
- Integrated into the application
- On-Demand & Live
- Smooth Streaming
- VC-1 & H.264



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0819
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Screen Capture

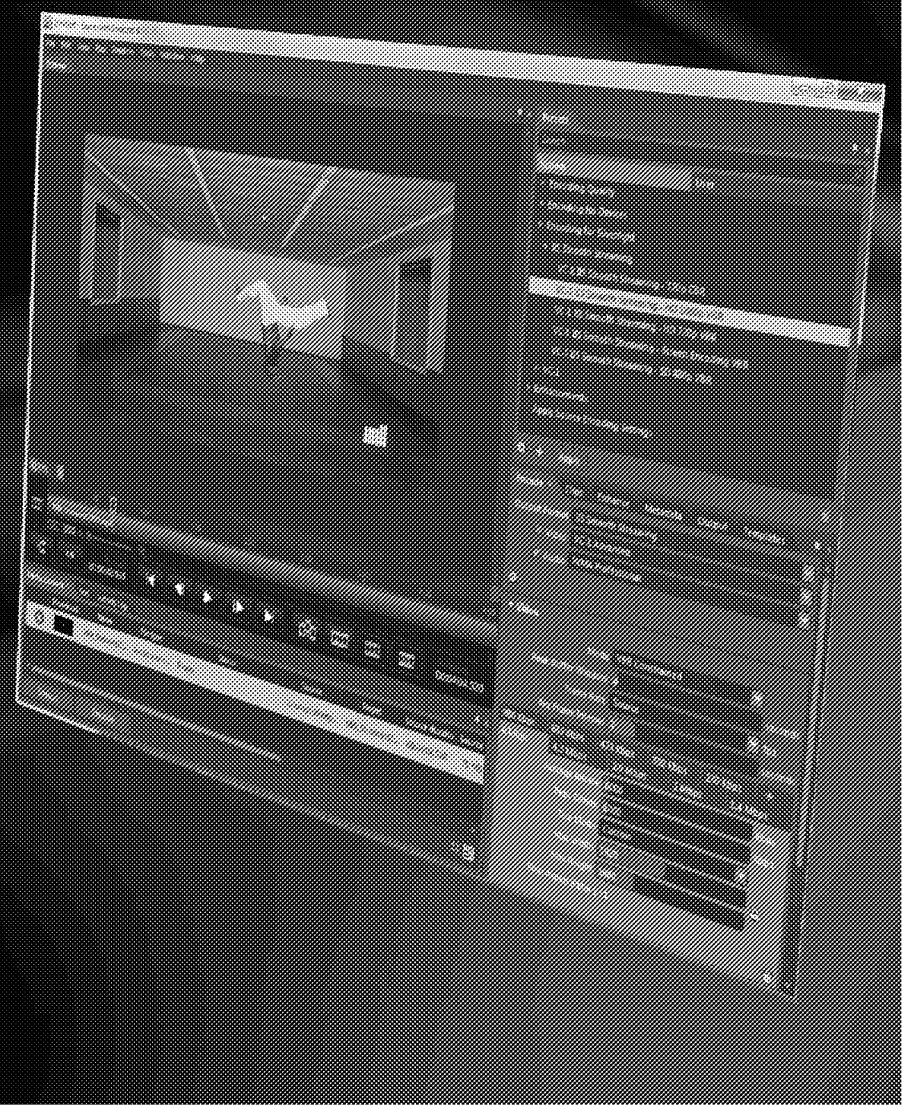
- **Registered Codec**
- **Full Object Model**
- **Capture Manager**
- **Zooming**
- **Mouse Tracking**



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0820
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

User Requests

- **Improved Closed Captioning**
- **Performance**
- **Azure Publishing**
- **Watch Folders (Sample)**



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0821
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Q&A
jlang@microsoft.com
http://social.expression.microsoft.com/Forums/en-US/silverlightnet/

Mobile Encoding Guidelines for Android™ Powered Devices

Addendum to *Video Encoding Cookbook and Profile Guidelines for the Adobe® Flash Platform*

By Maxim Levkov, Adobe Systems Inc.

Addendum
Table of Contents

1: Preface
2: Assumptions
2: Video and Audio Features in Flash for Mobile
3: Device Display Diversity
4: Network Connection Speed
5: Encoding Considerations
14: Video Encoding Considerations
17: Encoding Variants
23: Detailed AAC/AVC, Audio Video Settings
42: Conclusion

Preface

Mobile devices are quickly becoming a popular method of viewing media content. This rapid growth underscores the need for encoding guidelines to ensure that the content is optimized for reach and playback performance. It is not enough to simply deploy a single video player SWF with multi-bitrate content and expect a smooth playback experience on devices. For example, accommodations must be made for realities such as differing device capabilities, gesture interactions, screen orientation, and network connection speeds.

There are number of mobile platforms on the market today, but this document will focus specifically on Android™ powered devices. That being said, many of the general guidelines presented apply to other platforms as well. The continued evolution of Flash Player and the availability of increasingly powerful devices pushes video playback to the top of the list of most desired features on mobile networks today and well into the future.

This document is an addendum to the broader *Video Encoding Cookbook and Profile Guidelines for the Adobe Flash Platform* white paper. It would be beneficial to refer to that document, as well as the *Best Practices for Mobile Device Video Player Optimization* addendum also published separately.

Due to the wide variety of devices in the marketplace, there are two suggested approaches to encoding for mobile delivery. One approach is to target individual classes of mobile devices, optimizing encoding settings differently for each. The second approach is to service each of the diverse mobile phone devices through universally applicable encoding settings. Each of these approaches has advantages and disadvantages.

With the first approach, each of the device's unique capabilities can be taken into consideration and specifically encoded for, thus optimizing the viewing experience and perceptual quality. This approach is advantageous when the player's logic is set up to feed this specifically encoded content to that specific series of phone devices, or the aim is to service only an audience with a certain type of device, or even for local playback on device (e.g. download-to-own). The disadvantage of this approach is it's limited reach, mainly due to differing technical capabilities of devices and optimization of encoded content to reach only a specific device. For example, a mobile device that is able to play back video encoded for H.264 Main Profile at Level 3.1 (e.g. high end class) will not play efficiently or even be recognized on a mobile device that is able to only support H.264 Baseline Profile at Level 3.1 (e.g. medium end class), or a device that is able to support only H.264 Baseline Profile at Level 2.1 (e.g. low end class).

Hence, in the second approach, which takes a more universal style towards encoding, content is encoded using the lowest common set of parameters. The advantage of this approach is of course it's broad customer reach. The disadvantage is that is does not utilize each device's unique capabilities and maximum playback quality. This translates to reduced picture frame size and perceptual quality for all viewers but the lowest-powered devices. For example, if the group of phone devices includes high end (e.g. H.264 Main Profile at Level 3.1 support), medium (e.g. H.264 Baseline Profile at Level 3.1 support), and low end (e.g. H.264 Baseline Profile at Level 2.1 support), then the content would be encoded using the low end H.264 Baseline Profile at Level 2.1, since it is supported by all of the targeted devices.

Some examples of these classes of devices* are outlined in Table 1.

Class	Example Devices
High End	Motorola Droid X/Droid 2, HTC Evo, HTC Droid Incredible
Medium End	Google Nexus One, Motorola BACKFLIP™
Low End	Motorola Droid 1, Motorola CHARM™

* These devices are not exclusive representatives of their respective class of devices, and are presented for illustration purposes only.

Table 1
Sample Android powered devices and their classes.

Assumptions

Technical staff using this document should be skilled in the video coding technology field.

Quality control tools, viewing and listening conditions are tested and calibrated as described in *Video Encoding Cookbook and Profile Guidelines for the Adobe Flash Platform*, using the recommended test patterns and equipment.

Coded content is destined for appropriate compatible software and/or hardware decoders.

Coding software and hardware in use is functioning as stated.

Coding software and hardware supports at least some of the following coding elements mentioned throughout this document

Video Content	
Sizes	128x96 to 1920x1088
Frame Rates	23.976, 24, 25, 29.97, 30, 50, 59.94, 60fps or fraction thereof
Aspect Ratio	1.33, 1.78, 1.78 AN, 1.85, 2.35, including Letterbox and Pillarbox variants
Color Space	YUV 16 -235, Color Matrix 601 or 709
Video Sampling Structure	4:2:0
H.264 Video Content	
Coding Profiles	Baseline, Main
Coding Levels	1 through 3.2
At least one of the following muxing formats	F4V, MP4, MOV, 3GGP
At least one of the following audio coding formats	AAC LC, HEAAC v1, HEAAC v2

Video and Audio Features in Flash for Mobile

Flash Player provides acceleration features that aid in high-quality media playback on mobile devices.

Hardware Decoding of Audio

Hardware audio decoding uses the mobile device's hardware to accelerate audio decoding. Without hardware decoding, audio decoding is an intensive process that involves complex parsing and decompression operations requiring high CPU cycles and power consumption.

Hardware decoding provides equivalent functionality to software decoding in Flash Player with accelerated AAC audio decoding (Main, LC, HE/SBR profiles), and provides the following features:

- High-quality audio playback experience.
- Overall reduction in CPU usage, freeing up CPU cycles for other operations, which in turn improves performance and battery life.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0824
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Transparent to the user. If a hardware driver is not available or the audio codec format is not supported by the device hardware, Flash Player will fall back to software decoding.

Hardware Decoding of Video

Flash Player 10.1 introduced hardware-based H.264 video decoding to deliver smooth, high-quality video with minimal overhead across supported mobile devices and PCs.

Hardware accelerated rendering, GPU composition, and video hardware decoding combine to deliver high quality multimedia experiences on mobile devices with supported hardware. Benefits include:

- Offload tasks from the CPU to hardware, improving video playback performance, reducing system resource utilization, and preserving battery life.
- Deliver smooth, high-definition video with minimal overhead across devices.
- Use no CPU resources for video scaling. Audio and video can be decoded purely in hardware.
- Preserve battery life.

Device Display Diversity

Earlier, in the *preface*, device classification is mentioned as a means of identifying or grouping devices and the associated expectations for H.264 playback performance for these devices. It is useful to segment devices in this way not only for encoding purposes, but also for targeting display characteristics.

The following table highlights some examples of devices and their classes.

		Width x Height			pixel/inch	Mhz	
Motorola Droid X	High End	854 x 480	1.78:1 (16:9)	4.3"	228.3	1000	TI OMAP3630
Motorola Droid 2	High End	854 x 480	1.78:1 (16:9)	3.7"	264.7	1000	TI OMAP3620
Motorola Droid Pro	High End	480 x 320	1.5:1 (15:10)	3.1"	185.5	1000	TI OMAP3620
HTC Evo	High End	800 x 480	1.67:1 (15:9)	4.3"	217.4	1000	Qualcomm Snapdragon QSD8650
HTC Droid Incredible	High End	800 x 480	1.67:1 (15:9)	3.7"	252.1	1000	Qualcomm Snapdragon QSD8650
Samsung Galaxy S	High End	800x480	1.67:1 (15:9)	4.0"	235.1	1000	Samsung-Intrinsity 55PC110
Google Nexus One	Medium End	800 x 480	1.67:1 (15:9)	3.7"	252.1	998	Qualcomm Snapdragon QSD8250
HTC G2	Medium End	800 x 480		3.7	252.1	800	Qualcomm MSM7230
Motorola BackFlip	Medium End	480 x 320	1.5:1 (15:10)	3.1"	185.5	528	Qualcomm MSM7200A
Motorola Droid 1	Low End	800 x 480	1.67:1 (15:9)	3.7"	266.7	600	TI OMAP3430
Motorola Charm	Low End	320 x 240	1.33:1 (4:3)	2.6"	143.1	600	TI OMAP3410

Table 2

Examples of device classes.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0825
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Although all of the devices in the previous table are able to display H.264 encoded video, and they even exhibit some display similarities across different classes, each device still has unique combination of specifications such as screen size and pixel density. These similarities and differences make encoding content for mobile delivery difficult. What is optimally encoded for High End class display and processing capabilities (e.g. HTC Evo, Motorola Droid X, etc. at 800x480, 854x480) may not display on all Medium End class devices (e.g. Motorola Backflip at 480x320), and also on Low End class display and processing capabilities devices (e.g. Motorola Charm at 320x240). For medium and low end class devices, the processor will be taxed when trying to display content encoded at higher resolutions. For devices with higher display resolution, down-scaling can cause processing issues and scaling up the lower resolution content results in a noticeably lower quality image.

Taking a closer look at the display resolutions of the classified devices provides clear guidelines for encoding content that plays well across devices.

Figure 1 depicts the device resolution, from Table 2, in 1 to 1 pixel relationship.

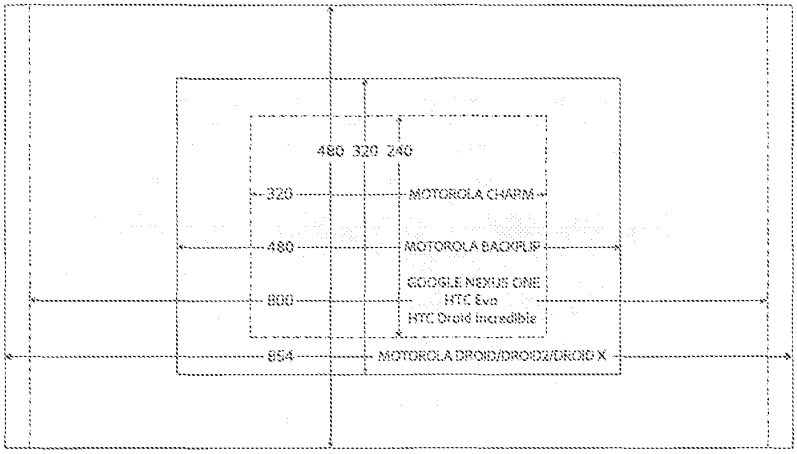


Figure 1
Device resolution from Table 2, in 1:1 pixel relationship.

What makes the resolution of the display fit the size of the display is the areal density of pixels within given physical display size in pixel per inch ratio. Although this pixel density is not something that can be controlled by the user or encoder, the picture size and picture aspect ratio of the encoded video is in the control of the encoder.

Network Connection Speed

Another crucial element in high quality playback on mobile devices is the end user's connection speed. For example, encoded content that is intended for delivery over broadband connections will not work efficiently over 3G networks due to their lower bandwidth capabilities. The end user connection that you decide to target will guide your encoding settings. Working within the boundaries of available network bandwidth for the targeted audience, while allowing for about 10-20% headroom for network fluctuations, is typically a good guideline to use as a starting point for optimal video playback. For example, if you determined through various means that the end users are only able to receive 500kbps of bandwidth, encoding for bandwidth of 800kbps will not yield a good playback experience. Although, 800kbps can still be delivered if using progressive delivery, but playback will stutter if the video data isn't being received fast enough, or when the device's buffer is full. So, while considering delivery to end user with 500kbps connection speed, encoding exactly at a bitrate of 500kbps will not provide a good experience either, as network conditions tend to be unpredictable. This is where the reduction of encoded data payload by 10-20% will play into effect. For 500kbps, it will be 400 – 450kbps effective total bitrate (i.e. audio and video combined).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0826
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The following table provides typical speed estimates of various networks, giving you a starting point for targeting bandwidth values.

EDGE	Peak bit-rates of up to 1Mbit/s and typical bit-rates of 400kbit/s can be expected.
3G	Typical current average download speed is between 600 Kbit/s – 1.4 Mbit/s.
4G	Typical current average download speed is between 3 Mbit/s – 6 Mbit/s.
HSPA+	Peak data rates up to 56 Mbit/s in the downlink in theory (up to 28 Mbit/s in existing services) and up to 22 Mbit/s in the uplink.
WiFi	Peak data rates up to 300 Mbit/s bi-directionally.

Source: Wikipedia.org. For general reference only; actual rates may vary by network provider.

Table 3

Speed estimates of various networks

Encoding Considerations

For more detailed explanation on how to prepare your content for highest quality playback, refer to *Video Encoding Cookbook and Profile Guidelines for the Adobe Flash Platform*, available for download at http://www.eventsadobe.com/cookbook.

The following ten guidelines should be followed when encoding content intended for mobile devices:

1. Keep the content in progressive output mode. Deinterlace whenever possible. Use Motion Compensated deinterlacing for best results. If not possible, use Motion Adaptive deinterlacing instead.
2. Use content that was deliberately shot for smaller screens for best user experience and fidelity. If not possible, scale with multi-tap filters (10 taps or more). The higher the tap the greater the sampling area (it is presumed that the filter is sophisticated enough to extract the necessary information to resample a better new image). High quality scaling filters utilize a higher sampling area from surrounding pixels (10 or more) and, thus, are able to derive higher quality output. Lower quality scaling filters (2-4 taps) create mediocre results, with images often being soft and blurry. Lower filter results may be acceptable for playback on the larger screen sizes of desktop computers, but will result in very poor picture quality on the smaller screen sizes of mobile devices. Lower end filters are faster than higher end filters due their smaller sampling area. Whenever possible and if available, use high quality scaling filters.
3. Maintain the aspect ratio of the original video.
4. Keep the frame size in multiples of 16 to avoid unnecessary performance degradation or CPU consumption. If not possible, resort only to multiples of 8, not 4.
5. If the transcoder/encoder supports two pass coding, use it. Typically a transcoder/encoder will use the first pass to index complex scenes and use the second pass for actual encoding. This process provides the best predictability for the coder and, consequentially, better output results. Depending on the complexity of the video, the perceptual video quality results from single pass to two pass encoding can vary as much as 10–30 percent, a valuable gain given the scarcity of bitrate resources.
6. Calibrate your Quality Assurance monitoring equipment for accurate representation of the output. (For more information on calibration and quality measurement, refer to the *Video Encoding Cookbook and Profile Guidelines for the Adobe Flash Platform* white paper.)
7. If the H.264 encoder supports "look_ahead" logic, use it. Specify at 1ames or greater, if coder permits.
8. Do not use content encoded at a larger frame size and then scale it down in the player. Create separate versions at the designated frame sizes at encoding or creation time. If higher frame size content is used, the device will attempt to scale it down, consuming excess CPU resources — as much as 40% in some cases. This slows performance of the device, reduces battery life, and degrades the overall playback experience.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0827
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

9. Do not use multiple slices mode in H.264. Instead, use 0 slices or 1 slice, or disable it all together. If slices are present in the video, the decoder will attempt to reproduce them while consuming unnecessary CPU resources.

10. If your source is interlaced and it needs to be scaled, deinterlace first then scale second.

Source

For best possible results, make sure that your source material is the highest quality available. It is strongly suggested that the original uncompressed source media file be used as an encoding source. Although this uncompressed media occupies considerably more disk space than a compressed (lossy) format, the encoding results are substantially better in quality than from a compressed source file — regardless of the level or method of compression. Because subsequent conversions will always exponentially decrease the sound and image quality, starting from a pristine and uncompromised video and audio file will ensure the best output for the final version.

Once you have a source to work with, properly formatting it for mobile delivery is as important as the quality of the source itself. This process involves resizing the original to a smaller frame size and compressing the video with mobile-friendly H.264 settings.

Picture Frame Sizes

Earlier, the section for *Device Display Diversity* detailed the impact frame sizes can have on performance of various classes of mobile devices. The highest display size for Android powered devices is 854x480 at 1.78:1 aspect ratio. The current highest display size on non-Android powered phone devices is 960x540 pixels (not including tablets)

When encoding for mobile devices it is important to minimize the impact on the processor by respecting the codec's optimal macroblock division of 16x16 pixels, while maintaining source picture aspect ratio. Additionally, it is important to ensure that the picture falls within the maximum display size of the targeted device or group of devices.

Most mobile devices allow displaying content either in landscape (horizontal) viewing mode or in portrait (vertical) viewing mode.

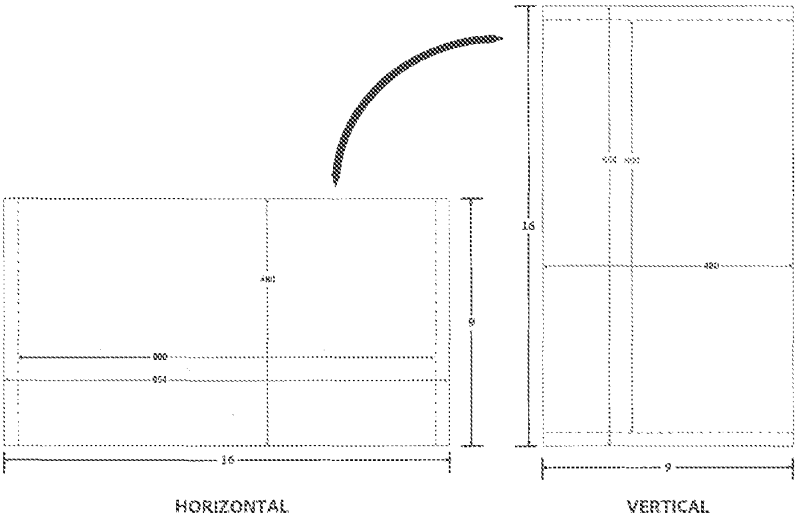


Figure 2

Horizontal and vertical screen sizes. This display flexibility creates a technical challenge in rendering the experience in vertical and horizontal positioning modes.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0828
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Horizontal (Landscape) Screen Positioning

If the maximum horizontal display sizes of your targeted devices are 854x480 and 800x480 there are two possible solutions for encoding optimal content with various picture aspect ratios.

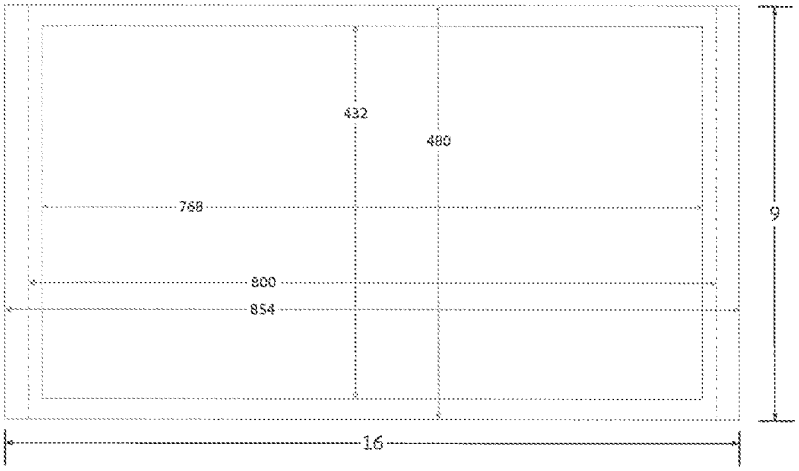


Figure 3

Optimal 16:9 aspect ratio conversion for screen height of 480 pixels

The frame size choice for this example, depicted in Figure 3, would be 768x432 because it is an exact 16x16 macroblock division and 1.777 (e.g. 16:9) aspect ratio. It also fits within the boundaries of the lowest resolution (800x480) of the two (854x480 and 800x480).

Solution one encodes the video with exact 16x16 macroblock division and exact matching aspect ratio (e.g. 16:9, 1.777:1) of the highest display size (e.g. 854x480) of the targets. The video is then scaled to fit the display, edge-to-edge, by the phone and player via full screen mode for the 854x480 size; and for the lower size of 800x480 the video is scaled to match the width of the player with the remaining height difference being filled with black bars on the top and bottom (letterbox).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0829
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

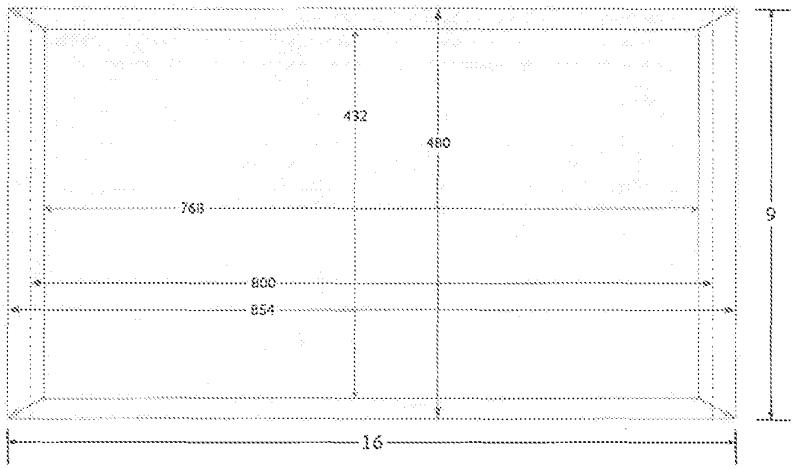


Figure 4

Scaling of 768x432 encoded picture size to fit 854x480 display size in "fit to screen mode," while maintaining 16:9 original picture aspect ratio. The picture is scaled by 11.198% from original picture size.

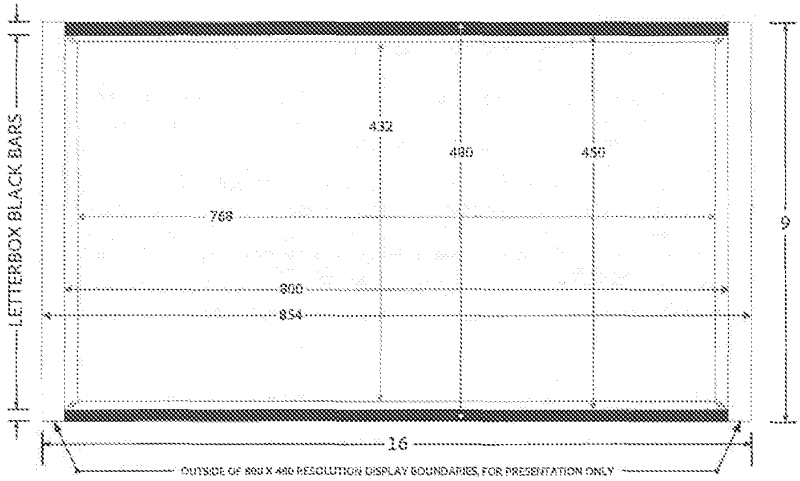


Figure 5

Scaling of picture encoded at 768x432 sized to fit 800x480 display size. Actual fitted picture size is 800x450 at 16:9 original picture aspect ratio, the remaining area of 30 pixels (480-450) is filled with black bars, therefore forming a letterbox. The picture is scaled by 4.167% from the original picture size (black bars excluded).

Solution two, like solution one, encodes the video with exact 16x16 macroblock division and exact matching aspect ratio (e.g. 16:9, 1.777:1) of the highest display size (e.g. 854x480) of the targets. The video is then displayed in native source resolution with pillar and letterbox black bars filling the balance of the screen resolution for either 854x480 or 800x480 size.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0830
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

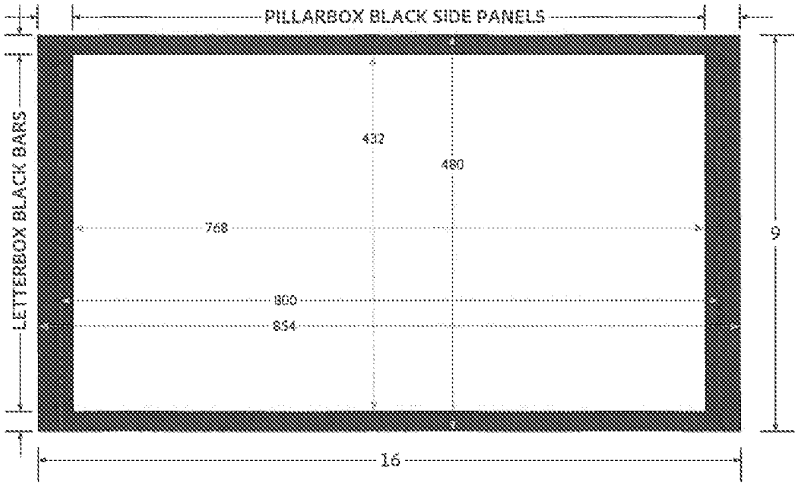


Figure 6

Native resolution of picture encoded at 768x432 encapsulated by the pillarbox black side panels and letterbox black bars without any scaling.

The presentation effect of this type is referred to as a postage-stamp in extreme cases when the picture is reduced by nearly half of the screen and black bars occupy about three quarters of the screen, but not in the case depicted in Figure 5. Although, in this case it avoids scaling and retains minimal impact on the processing, usually it is not very appealing to the viewer. Try to resort to this type of presentation only in cases where you are concerned about the processing impact of scaling up. Notice that scaling up has less impact on the processing performance versus scaling down, due to the use of an additive scheme in scaling up mode and a deductive scheme in scaling down mode.

Vertical (Portrait) Screen Positioning

When the viewer positions the device in such a way that it triggers vertical rendering of the screen, the video is scaled and formatted to play back horizontally within the height and width boundaries of the screen. Since vertical positioning of the screen (as shown in Figure 2) significantly reduces available space for viewing the active video window, there are three possible approaches:

- Simply scale the existing video to fit the maximum width and height to fit within the boundaries of the device, but at the cost of even more reduced objects in the video.
- Request the pre-encoded video that fits within the boundaries of the screen
- Fit the video to the screen dimensions without applying scaling.

In most cases vertical positioning of the screen will create a perfect case for use of video with 4:3 picture aspect ratio and multi-bitrate delivery.

Scenario one uses horizontally rendered video for vertically triggered positioning on the screen with scaling down to vertical boundaries of the device, as depicted in the following pictogram. However, this approach reduces the objects on already small screen to even smaller picture with reducing the objects further, thus making them harder to see. Additionally, such scaling process pushes the performance on the device to suboptimal state because it requires the use of unnecessary CPU resource for scaling that video. Also, the same amount of bits is piped down the network pipeline unnecessarily. For example, the video encoded for horizontal display at 16:9 picture aspect ratio, and sized to fit within the boundaries of the devices' screen resolution, such as 768x432 is encoded for 854x480 screen size, it will render at scaled 480 in width and corresponding height of 480x854 screen size in vertical position, or about 480x270 (at width x height) video resolution.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0831
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

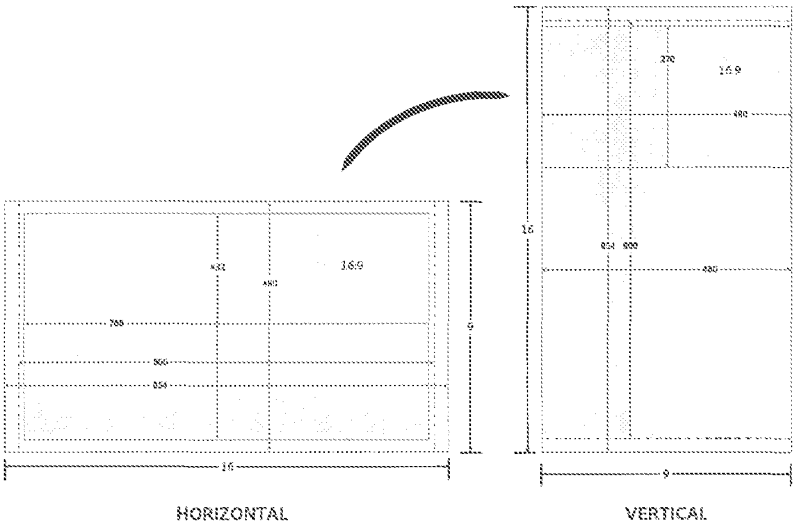


Figure 7
Fitting 16:9 aspect ratio video in vertical screen position from horizontal position.

Scenario two uses pre-encoded streams to fit the vertical positioned screen whenever the device is rotated to the vertical orientation. This approach is more complex than the first scenario, because it requires player logic to accommodate the different device position rendering whenever it is turned from horizontal to vertical state and vice versa. However, this scenario provides optimization on display of the video, reduced bit consumption due to reduced native video resolution requirements. This case also calls for use of 4:3 picture aspect ratio instead of 16:9, because it covers larger viewable area. For example, if video is encoded for horizontal display at 16:9 picture aspect ratio, and sized to fit within the boundaries of the devices' screen resolution, such as 768x432 (16:9) is encoded for 854x432 screen size, it will render at 480x360 (4:3) from a separate stream at request of a player to fit the vertical position of 480x854 screen size.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0832
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

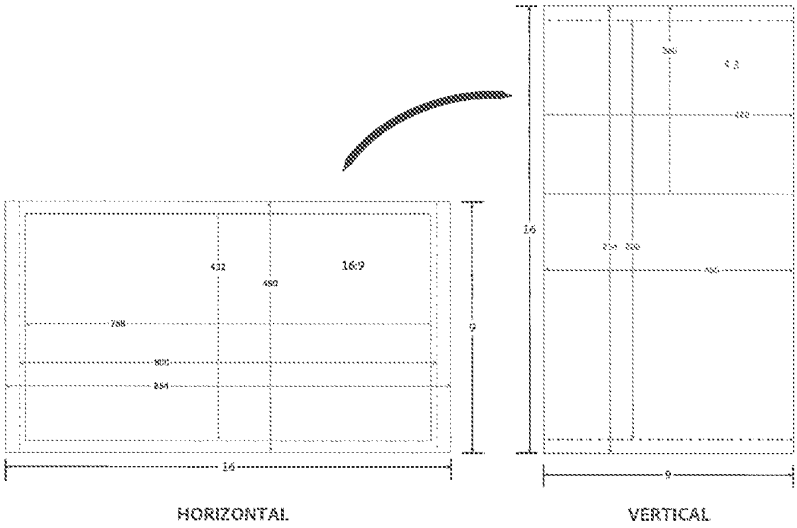


Figure 8

Fitting 4:3 aspect ratio video in vertical screen position from 16:9 horizontal position.

Scenario three is very similar to scenario two with one exception — it does not scale video to fit the screen, but rather keeps it in its natively encoded resolution.

Example Video Sizes for Horizontal and Vertical Orientations

The following tables provide example picture sizes and corresponding performance for an example device display with the following specifications:

- Horizontal orientation: Maximum screen size of 854x480 (16:9 aspect ratio)
- Vertical orientation: Maximum screen size of 480x854 (9:16 aspect ratio)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0833
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Video Size (Width x Height)	Aspect Ratio	Encoding	Quality
32 x 24	4:3	8x8	Good
64 x 48	4:3	16x16	Best
97 x 72	4:3	8x8	Good
128 x 96	4:3	16x16	Best
160 x 120	4:3	8x8	Good
192 x 144	4:3	8x8	Good
224 x 168	4:3	8x8	Good
256 x 192	4:3	16x16	Best
288 x 256	4:3	8x8	Good
320 x 240	4:3	16x16	Best
352 x 264	4:3	8x8	Good
384 x 288	4:3	16x16	Best
416 x 312	4:3	8x8	Good
448 x 336	4:3	16x16	Best
480 x 360	4:3	8x8	Good
128 x 72	16:9	16x16	Best
256 x 144	16:9	16x16	Best
384 x 216	16:9	16x16	Best
144x256	9:16	16x16	Best
216x384	9:16	8x8	Good
288x512	9:16	16x16	Best
360x640	9:16	8x8	Good
432x768	9:16	16x16	Best

Table 4

Video sizes for vertical orientation.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0834
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

[illegible]	[illegible]	[illegible]	[illegible]
32 x 24	4:3	8x8	Good
64 x 48	4:3	16x16	Best
97 x 72	4:3	8x8	Good
128 x 96	4:3	16x16	Best
160 x 120	4:3	8x8	Good
192 x 144	4:3	8x8	Good
224 x 168	4:3	8x8	Good
256 x 192	4:3	16x16	Best
288 x 256	4:3	8x8	Good
320 x 240	4:3	16x16	Best
352 x 264	4:3	8x8	Good
384 x 288	4:3	16x16	Best
416 x 312	4:3	8x8	Good
448 x 336	4:3	16x16	Best
480 x 360	4:3	8x8	Good
512 x 384	4:3	16x16	Best
544 x 408	4:3	8x8	Good
576 x 432	4:3	16x16	Best
608 x 456	4:3	8x8	Good
640 x 480	4:3	16x16	Best
672 x 504	4:3	8x8	Good
704 x 528	4:3	16x16	Best
736 x 552	4:3	8x8	Good
768 x 576	4:3	16x16	Best
128 x 72	16:9	16x16	Best
256 x 144	16:9	16x16	Best
384 x 216	16:9	16x16	Best
512 x 288	16:9	16x16	Best
640 x 360	16:9	16x16	Best
768 x 432	16:9	16x16	Best

Table 5

Video sizes for horizontal orientation.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0835
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Video Encoding Considerations

Encoding video for delivery to the ever-increasing variety of devices quickly becomes a complex task. Applying classifications to groups of devices can help simplify this process. This can be done by targeting profile and level settings, and assigning High, Medium and Low End classifications to devices.

Profile and Level Settings

With all the elements that go into encoding the optimal video, H.264 Profile and Level settings for the targeted device are two key elements that are often overlooked.

Level	Max Macroblocks/s	Max Frame Size	Max Video Bit Rate for Video Coding Layer (VCL)	Max Video Bit Rate for Video Coding Layer (VCL) for High Profile	Examples for High Profile (maximum stored frames)
1	1,485	99	64	80	128_96@30.9 (8) 176_144@15.0 (4)
1b	1,485	99	128	160	128_96@30.9 (8) 176_144@15.0 (4)
1.1	3,000	396	192	240	176_144@30.3 (9) 320_240@10.0 (3) 352_288@7.5 (2)
1.2	6,000	396	384	480	320_240@20.0 (7) 352_288@15.2 (6)
1.3	11,880	396	768	960	320_240@36.0 (7) 352_288@30.0 (6)
2	11,880	396	2,000	2,500	320_240@36.0 (7) 352_288@30.0 (6)
2.1	19,800	792	4,000	5,000	352_480@30.0 (7) 352_576@25.0 (6)
2.2	20,250	1,620	4,000	5,000	352_480@30.7(10) 352_576@25.6 (7) 720_480@15.0 (6) 720_576@12.5 (5)
3	40,500	1,620	10,000	12,500	352_480@61.4 (12) 352_576@51.1 (10) 720_480@30.0 (6) 720_576@25.0 (5)
3.1	108,000	3,600	14,000	17,500	720_480@80.0 (13) 720_576@66.7 (11) 1280_720@30.0 (5)
3.2	216,000	5,120	20,000	25,000	1,280_720@60.0 (5) 1,280_1,024@42.2 (4)

Table 6

Table excerpt from Wikipedia.com (http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC) on limits correlated to H.264 profile and level.

High End vs. Medium and Low End

Classification of devices as High, Medium, and Low End is mainly aimed at rating their performance capabilities for decoding video, within the mobile device market.

Low and Medium End devices, unlike the High End, have lower video processing resources and therefore encoding settings that target these devices should be adjusted accordingly. Constraints are mainly physical and limited to memory capacity to store video frames, CPU processing frequency (typically associated with speed), and whether or not a separate video and graphics processing unit is installed. Flash Player does not put any restrictions on the playback on the a particular device, and technically, you can play back your video aimed for the High End class of devices with more processing capabilities, but your playback performance will suffer. Unlike desktop playback, mobile video playback requires the need for device classifications so that the encoding is optimized to best service these targets.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0836
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Low and Medium End devices are best serviced by Baseline Profile with range of Level 2.0 to 3.1 with up to two reference frames on average.

High End devices, on the other hand, offer higher processing speed, higher screen resolution and pixel density, and access to higher network speeds. While these aspects offer a myriad of possibilities for users, the video encoding for targeting these devices is also more advanced. Of course, the video that is encoded for the Low and Medium End device range will still play back efficiently on High End devices, the inverse is not true when video is encoded for High End devices and played back on Low and Medium End devices.

High End devices are able to playback High and Main profile encoded video at Levels up to 3.2 with B-frame support for added encoding efficiency. However, despite such advances and expanded video processing functionality on High End mobile devices, the need to optimize for targeted devices' capabilities still exists, even at the High End device level.

Universal

Whether the encoding is aimed for the Low, Medium, or High End class of devices, the need to service various classes of devices individually may limit overall reach. When the need to reach wide span of devices exists, the approach could take one of two paths — one that is simplistic and second one that is more complex.

For the simplistic approach, the encoding is targeted using the lowest common profile and level (e.g. Baseline at Level 2.1, or Baseline at Level 3.0), respective of the profile and level restrictions. This provides a relatively simple approach to the encoding and player development and provides playback compatibility for many devices in the target range, however, it does not take advantage of the potential for higher quality and bitrate on High End devices.

For second approach is more complex. The encoding is targeted for all devices in range with optimization for each class of devices with specific profiles and levels, along with ample player heuristics with comprehensive detection logic. This approach may include a set of videos encoded for the Low End class with Baseline profile at Level 2.1, Medium End class with Baseline profile at level 3.0, and High End class with Main profile at Level 3.2. The player logic would be able to effectively detect the devices' capabilities and feed the appropriate video to that device. This approach is more complex to develop, but it offers the best possible playback on the widest range of devices. Also, this approach may include support for fall back to a universal, low quality video that can be played back on all devices, such as described in approach one.

Audio Encoding Considerations

Since video data consumes the majority of allotted bandwidth and must conform to the decoding capability of the device, audio encoding is typically relegated to secondary priority. This guide mainly focuses on the Advanced Audio Coding (AAC) codec, even though other audio codecs and coding modes are available. When bandwidth and processing power is limited on target devices, choosing the right AAC coding mode will give your viewers the best possible audio playback experience.

The AAC codec has evolved since its inception and is now supported by many devices through embedded hardware or natively via software in the operating system (e.g. Android OS). Flash Player takes advantage of these OS software and hardware capabilities. However, not all devices have sufficient resources to decode all of the variations of AAC coding modes. For instance, a device that would be able to decode AAC-LC (low complexity) coding would not necessarily be able to decode HE-AAC v2 SBR +PS coding (see Table 4 for acronym descriptions), because it requires more processing resources despite it being at least twice as efficient in coding as AAC-LC.

Just because there is hardware acceleration for decoding H.264 video, does not necessarily mean that hardware decoding of audio will be as efficient as video decoding. Depending on the complexity of your content and desired audio fidelity in playback on the device, you might want to selectively choose the desired bitrate vs. coding mode (see Table 7 for an overview of suggested encoding settings for various types of audio content).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0837
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Sampling Rate	22.5kHz	44.1kHz	44.1kHz
Bitrate	48	128	160
Codec	AAC LC	AAC LC	AAC LC

Table 7

Suggested encoding settings for various audio content types

For example, if your audio track is a narrative with no complex scenes (e.g. overlapping sounds, explosions, etc.) and overall audio/video bitrate is limited to 500kbits per second, it would be advisable to reduce the sampling rate to 22.5kHz at a bitrate of 48kbits per second using AAC LC coding mode, leaving 452kbits for video. If your audio track is a mix of different types of audio scenes, then a sampling rate of 44.1kHz at a bitrate of 128kbits per second using AAC LC coding mode would be better, leaving 372kbits for video. If your audio track consists of music, and the overall audio/video bitrate is limited to 500kbits per second, a sampling rate of 44.1kHz at bitrate of 160kbits per second using AAC LC coding mode would be advisable, leaving 340kbits for video.

Choosing AAC LC will provide adequate playback performance and compatibility across many devices. However, as you may have noticed, it does not always provide an optimal balance between video and audio data within the overall bitrate. In the case of relatively small bitrate targets, video suffers immensely due to the diminishing bitrate allocated for it in relation that allocated to audio.

As you determine your set of bitrates for multi-bitrate delivery, variations in audio bitrates and sampling rates will play a critical role in the smoothness of switching between streams as conditions change. For the best bitrate switching performance of audio, the audio sampling rate should be the same in all bitrate segments, because the decoder will not need to use processing cycles to adjust to the change.

Conversely, choosing High Efficiency AAC coding will further enhance your user's audio experience with reduced audio bitrate requirements, and will allow you to allocate more remaining bits for video. Choosing HE-AAC v2 (SBR) with 64kbits per second or 48kbits per second will help satisfy your bitrate requirement while allowing you to retain higher sampling rate than AAC-LC will need for the low bandwidth target. However, choosing HE-AAC v2 SBR+PS may require more processing cycles on the end user device, unless there is a hardware support for it. In this case, the software will use processing cycles, inhibiting synchronous performance while taking cycles needed for video decoding. So choose this mode sparingly. Typically, the audible quality of HE-AACv2 (SBR) at 64kbits per second would be equal to the audible quality of AAC-LC at 128kbits per second.

See http://en.wikipedia.org/wiki/HEAAC for any specific references to the HE-AAC audio standard.

Codec	bitrate (kbps)							
AAC LC		48	56	64	80	96	112	128	160
HE-AAC v1 (SBR)	40	48	56	64	80	96	112	128	
HE-AAC v2 (SBR)	40	48	56	64	80	96	112	128	
HE-AAC v2 (SBR + PS)	40	48							

The acronyms used in this table are:

AAC	Advanced Audio Coding
AAC-LC	Advanced Audio Coding Low Complexity
HE-AAC v1	High Efficiency Advanced Audio Coding version 1
HE-AAC v2 (SBR	High Efficiency Advanced Audio Coding version 2 (Spectral Band Replication)
HE-AAC v2 (SBR + PS)	High Efficiency Advanced Audio Coding (Spectral Band Replication + Parametric Stereo)

Table 8

Commonly used audio bitrates with AAC/HE-AAC encoding

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0838
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8,000 Hz	or	8kHz
11,000 Hz	or	11kHz
16,000 Hz	or	16kHz
22,500 Hz	or	22.5khz
24,000 Hz	or	24kHz
32,000 Hz	or	32kHz
44,100 Hz	or	44.1kHz
48,100 Hz	or	48kHz

Table 8
Commonly used audio frequencies

Testing

Once you have determined all of the necessary elements for your final output to mobile devices it is suggested that you inspect your selection with targeted devices before releasing it for general use --- and even before developing player logic. Playing streams, one at a time, using progressive or streaming delivery to make sure that you are seeing the desired output will ensure the playability of your media on the target device. This step will reduce troubleshooting time later, when you begin to transition to multi-bitrate encoding and delivery.

The article "Beginner's guide to deploying video on mobile devices with Flash" by Jens Loeffler (http://www.adobe.com/devnet/devices/articles/deploy_video_mobile.html) provides guidelines for developing and deploying basic players for mobile devices. Also refer to the *Best Practices for Mobile Device Video Player Optimization* addendum for additional guidelines and recommendations.

Encoding Variants

Earlier in the guidelines, various operating states for mobile devices were identified. The following figure (Figure 9) shows how supported device operating states and targeted performance can dictate encoding parameters. When you create sets of assets optimized for certain operating and performance states, the content must gracefully satisfy various other operating conditions to ensure the best playback experience for the end user.

Device operational state typically refers to one of two modes: landscape mode, where the device is positioned horizontally; and portrait mode, where the device is positioned vertically.

There are two ways to display a video in landscape mode: full screen and in-page:

- **Full screen** --- Video fills the screen entirely. Full screen mode can either be scaled up from a smaller size video (through forced full screen mode in player) or it can be a predefined size that is simply encoded to fit the full dimensions of the device screen. This separately encoded stream would be delivered whenever full screen mode is requested.
- **In-page view** --- Video does not fill the entire screen, sometimes remaining within the browser. While in horizontal orientation the video can be displayed in the in-page state where it simply remains within the boundaries of the browser, unscaled. Using in-page view is contingent upon the nature of the other elements that need to be displayed, and upon the performance state characteristics of the device.

Another operating state of the device is portrait mode. In this vertical orientation, there are three ways to display a video:

- **Pseudo-full screen** --- Video is fitted to left, right, and top edge of the screen, or left and right edge of the screen
- **In-page** --- Video is displayed within the boundaries of the web browser's page, un-scaled
- **Full screen** --- Video is scaled to fill the entire screen, edge to edge on all sides

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0839
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

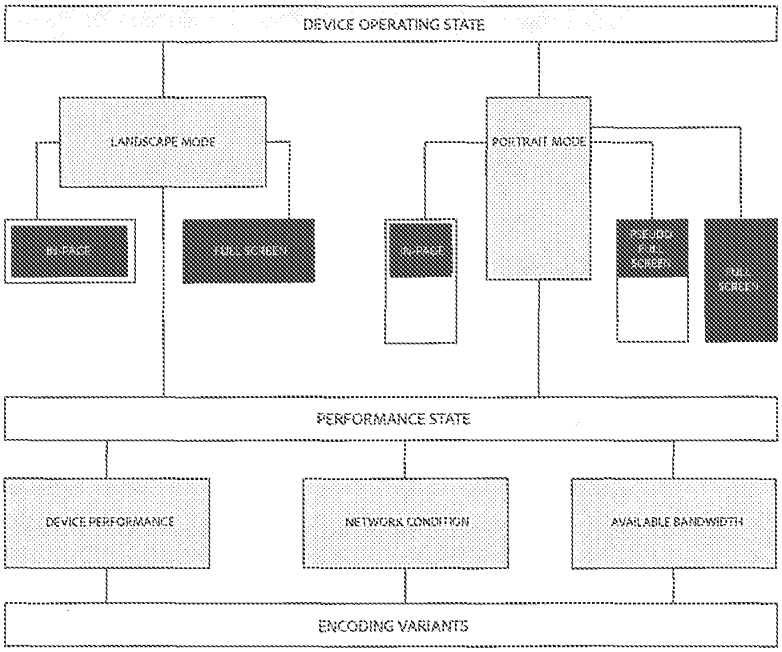


Figure 9

Various video playback states that dictate encoding variant requirements for devices

To summarize, the five viewing states of devices include:

1. Landscape in-page
2. Landscape full screen
3. Portrait in-page
4. Portrait pseudo full screen
5. Portrait full screen

The performance states depicted in Figure 9 consist of three elements that typically determine operating states. These elements are device performance, network conditions, and availability of bandwidth. Added to the operating state, the performance state helps to establish the number of renditions needed to gracefully react to changes in device performance, network conditions, and available bandwidth. Delivering an optimal viewing experience for devices requires specific coding variants and sets of encoded files.

Device performance is a key factor in determining optimal H.264 encoding settings.

Network conditions are another factor. Changes in network conditions determine the number of renditions needed and their bitrate proximity from each other. Video picture size can also be a variable when preparing multi-bitrate content.

Availability of bandwidth is a factor that is closely related to network conditions, but relates more to the maximum bandwidth available for video delivery on the device in any given moment (e.g while user is multitasking). The maximum bandwidth measurement helps to establish the minimum and maximum bitrate boundaries for the encoder to adhere to.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0840
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Given the above considerations, the following variants have been developed. These variants can be used as-is or modified to fit your specific needs.

Variants 1 through 5 limits the bandwidth use below 1000 Kbits/s, whereas variants 6 – 9 limits the top bandwidth cap at above 1000 Kbits/s through 2500 Kbits/s.

Each of the following variants highlights the recommended connection type and device class. Note that variants 1 through 5 target various ranges of devices and connections, so long as they are able to consume fewer than 1000 Kbits per second of bandwidth through available networks.

Variant 6 introduces bitrates above 1000 Kbits/s for devices in low, medium, and high classes of devices. The lower limit in this variant is moved to 250 Kbits per second.

Variant 7 mainly targets devices in medium and high classes and are able to consume 600 Kbits/s on the bottom of the bitrate range and 1864 Kbits/s at the top of the bitrate range.

Variant 8 combines a bottom bitrate range of 80 Kbits/s through a top bitrate range of 2500 Kbits/s for low, medium, and high device classes. Variant 9 targets the high end class of devices with a low bitrate range of 400 Kbits/s and high bitrate range of 1900 Kbits/s.

Variants 1 through 8 are based on Baseline profile, with levels ranging from 2.1 to 3.1. Variant 9 is based on Main profile.

Variant 1					
Video Quality					
Conn. Type	EDGE, 3G, 4G, WiFi				
Device Class	LOW, MEDIUM, HIGH				
	Video				
	Variant	Picture Size	H.264	HEAACv2	FPS
EDGE	1	128x72	56	24	80
EDGE	2	128x72	86	64	150
3G	3	256x144	186	64	250
3G	4	512x288	336	64	400
3G	5	512x288	536	64	600
4G, WiFi	6	640x360	736	64	800

Variant 2					
Video Quality					
Conn. Type	EDGE, 3G, 4G, WiFi				
Device Class	LOW, MEDIUM, HIGH				
	Video				
	Variant	Picture Size	H.264	HEAACv2	FPS
EDGE	1	128x72	56	24	80
EDGE	2	128x72	86	64	150
3G, 4G	3	256x144	186	64	250
3G, 4G	4	256x144	336	64	400
3G, 4G	5	512x288	536	64	600
WiFi	6	512x288	736	64	800

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0841
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Conn. Type	3G, 4G, WiFi				
Device Class	LOW, MEDIUM, HIGH				
			H.264	HEAACv2	
3G	1	256x144	202	48	250
3G	2	256x144	302	48	350
4G	3	256x144	402	48	450
4G	4	512x288	602	48	650
WiFi	5	512x288	702	48	850
WiFi	6	512x288	902	48	950

Conn. Type	3G, 4G, WiFi				
Device Class	LOW, MEDIUM, HIGH				
			H.264	HEAACv2	
3G	1	256x144	202	48	250
3G	2	512x288	302	48	350
4G	3	512x288	402	48	450
4G	4	512x288	602	48	650
WiFi	5	512x288	702	48	850
WiFi	6	512x288	902	48	950

Conn. Type	EDGE, 3G, 4G, WiFi				
Device Class	LOW, MEDIUM, HIGH				
			H.264	HEAACv2	
EDGE	1 (LOW)	128x72	56	24	80
EDGE	2 (LOW)	128x72	86	64	150
EDGE	3 (LOW)	256x144	186	64	250
3G	4 (MED)	512x288	336	64	400
3G	5 (MED)	512x288	536	64	600
4G, WiFi	6 (HIGH)	512x288	736	64	800

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0842
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Conn. Type	3G, 4G, WiFi				
Device Class	LOW, MED, HIGH				
	Video				
	Stream #	Screen Size	V	A	V
			H.264	HEAACv2	
3G	1 (LOW)	256x144	202	48	250
3G	2 (LOW)	256x144	352	48	400
4G	3 (MED)	512x288	552	48	600
4G	4 (MED)	512x288	752	48	800
WiFi	5 (HIGH)	512x288	952	48	1000
WiFi	6 (HIGH)	512x288	1252	48	1300

Conn. Type	3G, 4G, WiFi				
Device Class	MED,HIGH, HIGH				
	Video				
	Stream #	Screen Size	V	A	V
			H.264	HEAACv2	
3G, 4G	1	512x288	536	64	600
4G, WiFi	2	512x288	836	64	900
4G, WiFi	3	512x288	1000	64	1064
4G, WiFi	4	512x288	1200	64	1264
4G, WiFi	5	512x288	1500	64	1564
4G, WiFi	6	512x288	1800	64	1864

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0843
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Conn. Type	EDGE, 3G, 4G, WIFI				
Device Class	LOW, MEDIUM, HIGH				
	Video				
	Profile	Video Size		Audio	
			H.264	HEAACv2	
EDGE	1	128x72	56	24	80
EDGE	2	128x72	86	48	150
EDGE	3	256x144	202	48	250
EDGE	4	256x144	352	48	400
3G, 4G	5	512x288	552	48	600
3G, 4G	6	512x288	736	48	800
4G, WIFI	7	512x288	936	64	1000
4G, WIFI	8	512x288	1236	64	1300
4G, WIFI	9	512x288	1536	64	1600
4G, WIFI	10	512x288	1836	64	1864
4G, WIFI	11	512x288	2136	64	2200
4G, WIFI	12	512x288	2436	64	2500

Conn. Type	3G, 4G, WIFI				
Device Class	HIGH				
	Video				
	Profile	Video Size		Audio	
			H.264	HEAACv2	
3G	1	256x144	336	64	400
3G, 4G	2	512x288	536	64	600
4G, WIFI	3	512x288	836	64	900
4G, WIFI	4	512x288	936	64	1000
4G, WIFI	5	768x432	1136	64	1300
4G, WIFI	6	768x432	1536	64	1600
4G, WIFI	7	768x432	1836	64	1900

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0844
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Detailed AAC/AVC, Audio Video Settings

The following key defines the abbreviations found in the charts for Variants 1 - 9.

AVC/H.264	
SaS	Same as Source
FR	Frame Rate (in Frames per Second)
GT	GOP Type
KI/GOP Size	Keyframe Interval/GOP Size
ST	Scan Type
P	Progressive Scan Type
CA ECM	(Context Adaptive) Entropy Coding Mode
Pr@Lv	H.264 Profile and Level
IDR Frq.	Instantaneous Decode Refresh Frequency (I-frame picture frequency)
SS	Search Shape
wt P	Weighted P-picture mode
wt B	Weighted B-picture mode
8x8 Trsfm.	8x8 Transform
DBLK	Deblocking Filter On
ME	Motion Estimation Subpixel Mode
Ref Frms.	Reference Frames
#B Frm.	Number of B-frames
PM	Player Mode
FS	Full Screen
IP	In-page (e.g. regular mode)
SR	Search Range
B FDM	B Frame Direct Mode (prediction mode)
T	Temporal
H@4.2	High Profile at Level 4.2
M@4.2	Main Profile at Level 4.2
CBR/2pass	Either 2 pass in Constant Bitrate Mode or Capped Variable Bitrate Mode

HE-AAC	
SR (kHz)	Sample Rate (in Kilohertz)
BpS	Bits per Sample
HE AAC v2	High Efficiency Advanced Audio Coding version 2
SBR	Spectral Band Replication
PS	Parametric Stereo
# Ch	Number of Channels

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0845
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Variant 1

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

Conn. Type:	EDGE, 3G, 4G, WiFi										Device Class:	LOW, MEDIUM, HIGH												
		VIDEO Basic			Bitrate	H.264 Advanced																		
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	128x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	256x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	512x288	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	512x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	FS	640x360	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
		VIDEO Basic			Bitrate	H.264 Advanced																		
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	128x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	256x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	FS	480x270	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	FS	480x270	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	FS	480x270	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
		VIDEO Basic			Bitrate	H.264 Advanced																		
1	IP	96x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	96x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	192x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	384x288	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	384x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	IP	480x360	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0846
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1	HEAAC v2	Yes	No	24	32	16	2.0 (L,R)				
2	HEAAC v2	Yes	No	64	44.1	16	2.0 (L,R)				
3	HEAAC v2	Yes	No	64	44.1	16	2.0 (L,R)				
4	HEAAC v2	Yes	No	64	44.1	16	2.0 (L,R)				
5	HEAAC v2	Yes	No	64	44.1	16	2.0 (L,R)				
6	HEAAC v2	Yes	No	64	44.1	16	2.0 (L,R)				

1	80	1	
2	150	2	
3	250	3	
4	400	4	
5	600	5	
6	800	6	

Variant 2

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

Conn. Type:		EDGE, 3G, 4G, WiFi									Device Class:		LOW, MEDIUM, HIGH										
		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	128x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	256x144	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	512x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	FS	512x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	128x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	256x144	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	FS	480x270	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	FS	480x270	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	96x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	96x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	192x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	192x144	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	384x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	384x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0848
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1	HEAAC v2	Yes	No	24	32	16	270 (L,R)	1	80	1
2	HEAAC v2	Yes	No	64	44.1	16	270 (L,R)	2	150	2
3	HEAAC v2	Yes	No	64	44.1	16	270 (L,R)	3	250	3
4	HEAAC v2	Yes	No	64	44.1	16	270 (L,R)	4	450	4
5	HEAAC v2	Yes	No	64	44.1	16	270 (L,R)	5	600	5
6	HEAAC v2	Yes	No	64	44.1	16	270 (L,R)	6	800	6

Variant 3

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

1	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	256x144	SaS	P	302	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	402	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	512x288	SaS	P	602	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	512x288	SaS	P	702	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	FS	512x288	SaS	P	902	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

1	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	256x144	SaS	P	302	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	402	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	480x270	SaS	P	602	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	480x270	SaS	P	702	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	FS	480x270	SaS	P	902	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

1	IP	192x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	192x144	SaS	P	302	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	192x144	SaS	P	402	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	384x288	SaS	P	602	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	384x288	SaS	P	702	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	384x288	SaS	P	902	CBR/2pass	H.264	CAVLC	B@3.0	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0850
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1	HE-AAC v2	Yes	No	48	44.1	16	2.0 (L4)	1	250	1
2	HE-AAC v2	Yes	No	48	44.1	16	2.0 (L4)	2	350	2
3	HE-AAC v2	Yes	No	48	44.1	16	2.0 (L4)	3	450	3
4	HE-AAC v2	Yes	No	48	44.1	16	2.0 (L4)	4	650	4
5	HE-AAC v2	Yes	No	48	44.1	16	2.0 (L4)	5	750	5
6	HE-AAC v2	Yes	No	48	44.1	16	2.0 (L4)	6	950	6

Variant 4

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

Variant 4 VIDEO

VIDEO Basic | Bitrate | H.264 Advanced

1	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	512x288	SaS	P	302	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	512x288	SaS	P	402	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	512x288	SaS	P	602	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	512x288	SaS	P	702	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	FS	512x288	SaS	P	902	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

VIDEO Basic | Bitrate | H.264 Advanced

1	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	480x270	SaS	P	302	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	480x270	SaS	P	402	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	480x270	SaS	P	602	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	480x270	SaS	P	702	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	FS	480x270	SaS	P	902	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

VIDEO Basic | Bitrate | H.264 Advanced

1	IP	192x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	384x288	SaS	P	302	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	384x288	SaS	P	402	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	384x288	SaS	P	602	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	384x288	SaS	P	702	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	384x288	SaS	P	902	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0852
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1	HEAAC v.2	Yes	No	48	44.1	16	720 (L,R)	1	250	1	
2	HEAAC v.2	Yes	No	48	44.1	16	720 (L,R)	2	350	2	
3	HEAAC v.2	Yes	No	48	44.1	16	720 (L,R)	3	450	3	
4	HEAAC v.2	Yes	No	48	44.1	16	720 (L,R)	4	650	4	
5	HEAAC v.2	Yes	No	48	44.1	16	720 (L,R)	5	750	5	
6	HEAAC v.2	Yes	No	48	44.1	16	720 (L,R)	6	950	6	

Variant 5

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

[illegible]	[illegible]	[illegible]	[illegible]
Cone Type:	EDGE, 3G, 4G, WiFi	Device Class:	LOW, MEDIUM, HIGH
[illegible]	[illegible]	[illegible]	[illegible]

		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	128x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	512x288	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	512x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	512x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	128x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	512x288	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	512x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	512x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	96x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	96x72	SaS	P	86	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	192x144	SaS	P	186	CBR/2pass	H.264	CAVLC	B@2.2	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	384x288	SaS	P	336	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	384x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	384x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0854
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1	HEAAC v2	Yes	No	24	32	16	270 (1.8)	1	80	1
2	HEAAC v2	Yes	No	64	44.1	16	270 (1.8)	2	150	2
3	HEAAC v2	Yes	No	64	44.1	16	270 (1.8)	3	250	3
4	HEAAC v2	Yes	No	64	44.1	16	270 (1.8)	4	400	4
5	HEAAC v2	Yes	No	64	44.1	16	270 (1.8)	5	600	5
6	HEAAC v2	Yes	No	64	44.1	16	270 (1.8)	6	850	6

Variant 6

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

Video Basic					Bitrate	H.264 Advanced																		
1	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	256x144	SaS	P	336	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	512x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	512x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	512x288	SaS	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	IP	512x288	SaS	P	1236	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
Video Basic					Bitrate	H.264 Advanced																		
1	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	256x144	SaS	P	336	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	480x270	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	480x270	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	480x270	SaS	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	IP	480x270	SaS	P	1236	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
Video Basic					Bitrate	H.264 Advanced																		
1	IP	192x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	192x144	SaS	P	336	CBR/2pass	H.264	CAVLC	B@2.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	384x288	SaS	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	384x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	384x288	SaS	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	IP	384x288	SaS	P	1236	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0856
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1	HEAAC v2	Yes	No	48	44.1	16	2 (5.1)	1	250	1
2	HEAAC v2	Yes	No	64	44.1	16	2 (5.1)	2	400	2
3	HEAAC v2	Yes	No	64	44.1	16	2 (5.1)	3	600	3
4	HEAAC v2	Yes	No	64	44.1	16	2 (5.1)	4	800	4
5	HEAAC v2	Yes	No	64	44.1	16	2 (5.1)	5	1000	5
6	HEAAC v2	Yes	No	64	44.1	16	2 (5.1)	6	1300	6

Variant 7

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

Conn. Type:		3G, 4G, WiFi											Device Class:		MEDIUM, HIGH									
		VIDEO Basic			Bitrate		H.264 Advanced																	
Index	Type	Resolution	Sar	Scan	Bitrate	Mode	Profile	Entropy	Level	Ref	Slice	Structure	Closure	Ref	B	Size	Trans	Intra	Deblock	Cabac	Weight	Accuracy	Ref	
1	IP	512x288	5x5	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	512x288	5x5	P	836	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	512x288	5x5	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	512x288	5x5	P	1136	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	512x288	5x5	P	1536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	IP	512x288	5x5	P	1836	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
		VIDEO Basic			Bitrate		H.264 Advanced																	
Index	Type	Resolution	Sar	Scan	Bitrate	Mode	Profile	Entropy	Level	Ref	Slice	Structure	Closure	Ref	B	Size	Trans	Intra	Deblock	Cabac	Weight	Accuracy	Ref	
1	IP	480x270	5x5	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	480x270	5x5	P	836	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	480x270	5x5	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	480x270	5x5	P	1136	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	480x270	5x5	P	1536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	IP	480x270	5x5	P	1836	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
		VIDEO Basic			Bitrate		H.264 Advanced																	
Index	Type	Resolution	Sar	Scan	Bitrate	Mode	Profile	Entropy	Level	Ref	Slice	Structure	Closure	Ref	B	Size	Trans	Intra	Deblock	Cabac	Weight	Accuracy	Ref	
1	IP	384x288	5x5	P	536	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
2	IP	384x288	5x5	P	836	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
3	IP	384x288	5x5	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
4	IP	384x288	5x5	P	1136	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
5	IP	384x288	5x5	P	1536	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	
6	IP	384x288	5x5	P	1836	CBR/2pass	H.264	CAVLC	B@3.1	0	T	Fixed, 5x5 x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1	

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0858
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1	HEAAC v2	Yes	No	48	44.1	16	2/0 (L,R)		1	600	1	
2	HEAAC v2	Yes	No	64	44.1	16	2/0 (L,R)		2	900	2	
3	HEAAC v2	Yes	No	64	44.1	16	2/0 (L,R)		3	1200	3	
4	HEAAC v2	Yes	No	64	44.1	16	2/0 (L,R)		4	1300	4	
5	HEAAC v2	Yes	No	64	44.1	16	2/0 (L,R)		5	1600	5	
6	HEAAC v2	Yes	No	64	44.1	16	2/0 (L,R)		6	1900	6	

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0859
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Variant 8

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

[illegible]																							
Conn. Type		EDGE, 3G, 4G, WIFI										Device Class		LOW, MEDIUM, HIGH									
		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	128x72	SaS	P	102	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	256x144	SaS	P	352	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	512x288	SaS	P	552	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	512x288	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
7	IP	512x288	SaS	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
8	IP	512x288	SaS	P	1236	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
9	IP	512x288	SaS	P	1536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
10	IP	512x288	SaS	P	1836	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
11	IP	512x288	SaS	P	2136	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
12	IP	512x288	SaS	P	2436	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
		VIDEO Basic			Bitrate		H.264 Advanced																
1	IP	128x72	SaS	P	56	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	128x72	SaS	P	102	CBR/2pass	H.264	CAVLC	B@2.1	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	256x144	SaS	P	202	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	256x144	SaS	P	352	CBR/2pass	H.264	CAVLC	B@2.2	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	480x270	SaS	P	552	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	480x270	SaS	P	736	CBR/2pass	H.264	CAVLC	B@3.0	0	F	Fixed, SaS x 2	Closed	1	2	8x8	64	Yes	No	No	Yes	1/4 Pel	1
7	IP	480x270	SaS	P	936	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
8	IP	480x270	SaS	P	1236	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
9	IP	480x270	SaS	P	1536	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
10	IP	480x270	SaS	P	1836	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
11	IP	480x270	SaS	P	2136	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
12	IP	480x270	SaS	P	2436	CBR/2pass	H.264	CAVLC	B@3.1	0	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0860
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Mobile Encoding Guidelines for Android Powered Devices White Paper 39

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0861
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Variant 9

Use Case:

- Fixed Screen Sizes
- Fixed Page Sizes
- Variable Bitrates

		VIDEO Basic			Bitrate		H.264 Advanced																			
1	IP	512x288	SaS	P	336	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	512x288	SaS	P	536	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	512x288	SaS	P	836	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	512x288	SaS	P	936	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	768x432	SaS	P	1136	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	768x432	SaS	P	1536	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
7	IP	768x432	SaS	P	1836	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
		VIDEO Basic			Bitrate		H.264 Advanced																			
1	IP	480x270	SaS	P	336	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	480x270	SaS	P	536	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	480x270	SaS	P	836	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	480x270	SaS	P	936	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	480x270	SaS	P	1136	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	480x270	SaS	P	1536	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
7	IP	480x270	SaS	P	1836	CBR/2pass	H.264	CABAC	M@3.1	2	F	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
		VIDEO Basic			Bitrate		H.264 Advanced																			
1	IP	384x288	SaS	P	536	CBR/2pass	H.264	CAVLC	M@3.1	2	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
2	IP	384x288	SaS	P	536	CBR/2pass	H.264	CAVLC	M@3.1	2	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
3	IP	384x288	SaS	P	836	CBR/2pass	H.264	CAVLC	M@3.1	2	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
4	IP	384x288	SaS	P	936	CBR/2pass	H.264	CAVLC	M@3.1	2	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
5	IP	576x432	SaS	P	1136	CBR/2pass	H.264	CAVLC	M@3.1	2	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
6	IP	576x432	SaS	P	1536	CBR/2pass	H.264	CAVLC	M@3.1	2	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1
7	IP	576x432	SaS	P	1836	CBR/2pass	H.264	CAVLC	M@3.1	2	T	Fixed, SaS x 2	Closed	1	4	8x8	64	Yes	No	No	Yes	1/4 Pel	1

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0862
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Profile	Audio Codec	Video Codec	Resolution	Frame Rate	Video Bitrate	Audio Bitrate	Audio Sample Rate	Audio Channels	Key Frame Interval
1	HEAAC v2	Yes	No	48	44.1	16	2/3 (1, 2)	1	1
2	HEAAC v2	Yes	No	64	44.1	16	2/3 (1, 2)	2	2
3	HEAAC v2	Yes	No	64	44.1	16	2/3 (1, 2)	3	3
4	HEAAC v2	Yes	No	64	44.1	16	2/3 (1, 2)	4	4
5	HEAAC v2	Yes	No	64	44.1	16	2/3 (1, 2)	5	5
6	HEAAC v2	Yes	No	64	44.1	16	2/3 (1, 2)	6	6
7	HEAAC v2	Yes	No	64	44.1	16	2/3 (1, 2)	7	7

Profile	Video Bitrate
1	400
2	600
3	900
4	1200
5	1500
6	1600
7	1900

Conclusion

As mobile devices gain more and more integration into people's lives, media consumption and expectations of playback quality will continue to grow. It is the responsibility of the encoding professional, along with the developer, to optimize content for the best possible viewing experience. It is the aim of this document to provide workable guidelines and best practices for achieving this goal on Android-powered devices. As these devices evolve this document will be updated and amended.

Acknowledgements

The author would like to thank the following people for their valuable assistance in the development of this document:

Lisa Larson-Kelley for excellent editing, creative insight, and for outstanding day to day cooperation on every aspect of this cookbook.

Ashley Still for taking a chance with me on this unprecedented endeavor, for her leadership, and for the opportunity itself.

Pritham Shetty for entrusting me to do this project.

David Devisser for active participation and help resolving day to day issues.

Desiree Motamedi for paving the way for the endearing presentations and helping to acquire great content.

Jens Loeffler for unprecedented day-to-day cooperation on the many tasks this project has entailed.

Victor Steinberg of VideoQ for meticulous technical oversight, lots of time and extraordinary cooperation, excellent ideas, and great engineering in creation of specialized dynamic test patterns.

Matthew Kaufman for straightforward feedback and unparalleled insight into the inner workings of Flash, and for help in putting together the Encoding HTTP Delivery for Flash addendum section of the cookbook.

Kevin Towes for keeping me in-tune with latest developments, and then some.

Srinivas Manapragada for helping to put forth the Encoding HTTP Delivery for Flash addendum section of the cookbook.

Abhinav Kapoor, Vishy Swaminathan, Kevin Streeter, and Slavik Lozben for insight on the inner workings of Flash and for their feedback on various sections of the cookbook.

Special appreciation to:

ArtBeats for providing excellent content and timely delivery.

VideoQ for making variety of special test patterns, timely availability, and unquestionable general technical oversight.

Hillman Curtis for providing creative content.

© 2010 Adobe Systems Incorporated. All rights reserved.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

This article is intended for US audiences only.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe and the Adobe logo, Flash, and Flash Player are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

12/10

Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704
USA
www.adobe.com

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0864
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

SILVER: Simplifying Video Editing With Metadata

A. Chris Long[1], Juan Casares, Brad A. Myers, Rishi Bhatnagar, Scott M. Stevens, Laura Dabbish, Dan Yocum, and Albert Corbett
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 3527
chrislong@acm.org

ABSTRACT
Digital video is becoming increasingly ubiquitous. However, editing video remains difficult for several reasons: it is a time-based medium, it has dual tracks of audio and video, and current tools force users to work at the smallest level of detail. Based on interviews with professional video editors, we developed a video editor, called Silver, that uses metadata to make digital video editing more accessible to novices. To help users visualize video, Silver provides multiple views with different semantic content and at different levels of abstraction, including storyboard, editable transcript, and timeline views. Silver offers smart editing operations that help users resolve the inconsistencies that arise because of the different boundaries in audio and video.

Keywords
Digital video editing, multimedia authoring, metadata, Silver, Informedia

INTRODUCTION
Digital video is becoming increasingly ubiquitous. As technology and content become available, users will shift from passive spectators to active creators of video. Forrester Research predicts that by 2005, 92% of online consumers will create personal multimedia content at least once a month [4]. However, although many exciting research projects are investigating how to search, visualize, and summarize digital video, there is little work on new ways to support the use of the video beyond just playing it.

This is unfortunate, because video editing has several unique challenges not found with other media. One is that digital video is a time-based medium, which hampers browsing and skimming video.

Another challenge for editing video is that it is a dual medium, consisting of an audio track in addition to the video track. These tracks must be kept synchronized, but the user must also be able to overlay them when desired, for example during transitions from one shot to another. Further, when a shot is cut for use elsewhere, the user must be able to disentangle overlaid audio and video.

Copyright is held by the author/owner(s).
CHI 2003, April 5-10, 2003, Ft. Lauderdale, Florida, USA.
ACM 1-58113-637-4/03/0004

A third problem is that the syntactic units that users want to edit are shots of video and words or sentences of audio, but current tools require users to examine video at the individual frame level and audio using a waveform. To perform most editing operations, the user must manually pinpoint specific frames, which may involve zooming and many repetitions of fast-forward and rewind operations.

These problems make editing video a difficult, tedious, and error-prone activity. Commercially available tools, such as Adobe Premiere [1] and Apple iMovie [2], allow creation of high-quality video, but they do not adequately address the issues raised above, which makes them harder to use, especially for novices.

Based on observations of an interviews with video professionals, we created Silver, a system that uses metadata and video-specific interaction techniques to make video editing easier [3]. The key innovations in the Silver editor include: providing an editable transcript view; coordinating the selection across all views including partial selections and different selections in audio and video; and smart editing through smart snap, smart selection and smart cut and paste.

Silver simplifies video editing primarily by using metadata such as shot breaks, thumbnail images, and clip titles—all of which are obtained from the Informedia digital library [5]—to enhance the display of and interaction with the video composition, and by providing multiple, synchronized views of the video composition.

SILVER INTERFACE
An overview of the Silver interface, with all the views it offers, is shown in Figure 1. At the highest level of

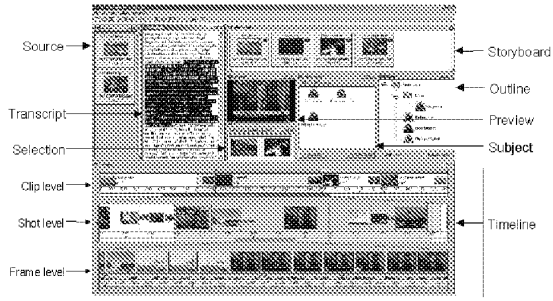


Figure 1 Overview of the Silver interface

[1]Current affiliation: ECE Department, Carnegie Mellon University.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0865
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

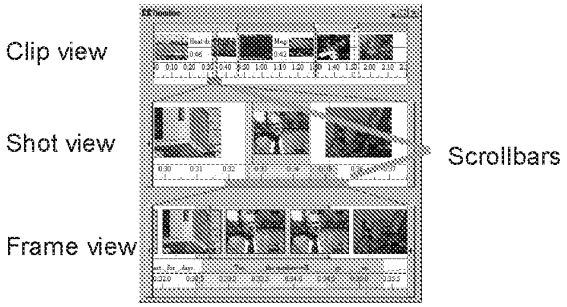


Figure 5 Timeline window.

abstraction, the composition is composed of video *clips* (i.e., video from a single file). Clips that have been previously found are shown in the source window. Clips can be organized in the outline and subject views, and can be put in the composition by dragging into in the storyboard window, where they can also be rearranged. When part of the composition is selected, the selection window shows the starting and ending frames.

The main part of the Silver interface is the hierarchical timeline, which shows video at three scales: clip, shot, and frame (see Figure 2). The clip and shot levels show the video composition as continuous segments from a single source file or camera, respectively. Each clip is displayed as a representative frame and, if space allows, its end frame, title, and duration. Each shot is displayed using a representative frame.

Users control how much is displayed in the shot and frame levels using a sophisticated scrollbar at the bottom of the clip level. In addition to the usual dragging behavior, the scrollbar may also be resized to zoom the shot level in or out. It also contains a miniature version of the shot level scrollbar, which shows the position of the frame level within the shot level. As in the clip level, there is an augmented scrollbar at the bottom of the shot level used to scroll and zoom the frame level.

The frame level displays the individual frames of the video, so the user can quickly find particular cut points. The frame level addresses the dual medium issue by displaying the transcript, including the start and end times of every word.

An important feature of the timeline is smart selection and editing. Silver uses metadata to ease editing with semantically meaningful units, for example, by snapping selection to shot boundaries and audio selection to word boundaries (see Figure 2). The user can disable snapping for manual selection if desired.

Another key feature of the timeline is that it shows different representations together, allowing the user to see the clip boundaries, the time, samples of frames, the cuts, the transcript, annotations, etc.

Silver also displays a separate, synchronized view of the transcript in another window (see Figure 3). Selection in

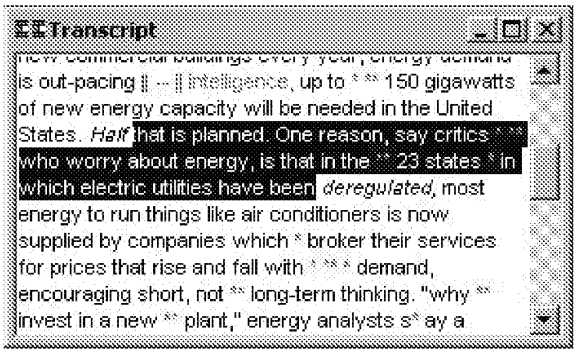


Figure 4 Transcript window. Blue vertical bars ("‖") show clip breaks. Dashes show gaps in audio. The word "intelligence" was cut in the middle. The green stars represent noise or recognition errors. At the edges of the selection, *half* and *deregulated* are partially selected.

the transcript and timeline are synchronized, and editing the transcript edits the video composition.

Silver allows audio and video to be selected independently. However, inserting or deleting unequal amounts of audio and video creates a hole in one of the tracks. When this happens, Silver notifies the user and provides options for closing the gap.

CONCLUSIONS

Silver is designed to make high-quality video editing easier, and demonstrates several techniques for overcoming difficulties of authoring in this domain. Specifically, it uses coordinated multi-level views, synchronized selection, and a textual transcript.

Silver shows that video metadata can be used to create an advanced interface for aiding novice users in the performance of otherwise difficult or tedious tasks. As more metadata becomes available, the smart editing approach of Silver can be used to improve video editing capabilities even further.

ACKNOWLEDGMENTS

The Silver Project is funded in part by the National Science Foundation under Grant No. IIS-9817527, as part of the Digital Library Initiative-2. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

REFERENCES

1. Apple Computer, Inc. iMovie. http://www.apple.com/imovie/.

2. Adobe Systems Incorporated. Adobe Premiere. http://www.adobe.com/products/premiere/main.html.

3. Casares, J., Long, A. C., Myers, B. A., Bhatnagar, R., Stevens, S. M., Dabbish, L., Yocum, D., and Corbett, A. "Simplifying Video Editing Using Metadata." In *Proceedings of DIS2002* (London, UK, June 2002), p. 159–166.

4. Schwartz, J., Rhinelander, T. and Dorsey, M., "Personal Rich Media Takes Off", The Forrester Report, Forrester Research Inc., October 2000.

5. Wactlar, H.D., et al., "Lessons learned from building a terabyte digital video library.", IEEE Computer, 1999, 32(2): pp. 66-73.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0866
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

"EA IFF 85" Standard for Interchange Format Files

Document Date: January 14, 1985
From: Jerry Morrison, Electronic Arts
Status of Standard: Released and in use

1. Introduction

Standards are Good for Software Developers

As home computer hardware evolves to better and better media machines, the demand increases for higher quality, more detailed data. Data development gets more expensive, requires more expertise and better tools, and has to be shared across projects. Think about several ports of a product on one CD-ROM with 500M Bytes of common data!

Development tools need standard interchange file formats. Imagine scanning in images of "player" shapes, moving them to a paint program for editing, then incorporating them into a game. Or writing a theme song with a Macintosh score editor and incorporating it into an Amiga game. The data must at times be transformed, clipped, filled out, and moved across machine kinds. Media projects will depend on data transfer from graphic, music, sound effect, animation, and script tools.

Standards are Good for Software Users

Customers should be able to move their own data between independently developed software products. And they should be able to buy data libraries usable across many such products. The types of data objects to exchange are open-ended and include plain and formatted text, raster and structured graphics, fonts, music, sound effects, musical instrument descriptions, and animation.

The problem with expedient file formats typically memory dumps is that they're too provincial. By designing data for one particular use (e.g. a screen snapshot), they preclude future expansion (would you like a full page picture? a multi-page document?). In neglecting the possibility that other programs might read their data, they fail to save contextual information (how many bit planes? what resolution?). Ignoring that other programs might create such files, they're intolerant of extra data (texture palette for a picture editor), missing data (no color map), or minor variations (smaller image). In practice, a filed representation should rarely mirror an in-memory representation. The former should be designed for longevity; the latter to optimize the manipulations of a particular program. The same filed data will be read into different memory formats by different programs.

The IFF philosophy: "A little behind-the-scenes conversion when programs read and write files is far better than NxM explicit conversion utilities for highly specialized formats."

So we need some standardization for data interchange among development tools and products. The more developers that adopt a standard, the better for all of us and our customers.

Here is "EA IFF 1985"

Here is our offering: Electronic Arts' IFF standard for Interchange File Format. The full name is "EA IFF 1985". Alternatives and justifications

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0867
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

are included for certain choices. Public domain subroutine packages and utility programs are available to make it easy to write and use IFF-compatible programs.

Part 1 introduces the standard. Part 2 presents its requirements and background. Parts 3, 4, and 5 define the primitive data types, FORMs, and LISTs, respectively, and how to define new high level types. Part 6 specifies the top level file structure. Appendix A is included for quick reference and Appendix B names the committee responsible for this standard.

References

American National Standard Additional Control Codes for Use with ASCII, ANSI standard 3.64-1979 for an 8-bit character set. See also ISO standard 2022 and ISO/DIS standard 6429.2.

Amiga[tm] is a trademark of Commodore-Amiga, Inc.

C, A Reference Manual, Samuel P. Harbison and Guy L. Steele Jr., Tartan Laboratories. Prentice-Hall, Englewood Cliffs, NJ, 1984.

Compiler Construction, An Advanced Course, edited by F. L. Bauer and J. Eickel (Springer-Verlag, 1976). This book is one of many sources for information on recursive descent parsing.

DIF Technical Specification (c)1981 by Software Arts, Inc. DIF[tm] is the format for spreadsheet data interchange developed by Software Arts, Inc.
DIF[tm] is a trademark of Software Arts, Inc.

Electronic Arts[tm] is a trademark of Electronic Arts.

"FTXT" IFF Formatted Text, from Electronic Arts. IFF supplement document for a text format.

Inside Macintosh (c) 1982, 1983, 1984, 1985 Apple Computer, Inc., a programmer's reference manual.
Apple(R) is a trademark of Apple Computer, Inc.
Macintosh[tm] is a trademark licensed to Apple Computer, Inc.

"ILBM" IFF Interleaved Bitmap, from Electronic Arts. IFF supplement document for a raster image format.

M68000 16/32-Bit Microprocessor Programmer's Reference Manual(c) 1984, 1982, 1980, 1979 by Motorola, Inc.

PostScript Language Manual (c) 1984 Adobe Systems Incorporated.
PostScript[tm] is a trademark of Adobe Systems, Inc.
Times and Helvetica(R) are trademarks of Allied Corporation.

InterScript: A Proposal for a Standard for the Interchange of Editable Documents (c)1984 Xerox Corporation.
Introduction to InterScript (c) 1985 Xerox Corporation.

2. Background for Designers

Part 2 is about the background, requirements, and goals for the standard.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0868
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

It's geared for people who want to design new types of IFF objects. People just interested in using the standard may wish to skip this part.

What Do We Need?

A standard should be long on prescription and short on overhead. It should give lots of rules for designing programs and data files for synergy. But neither the programs nor the files should cost too much more than the expedient variety. While we're looking to a future with CD-ROMs and perpendicular recording, the standard must work well on floppy disks.

For program portability, simplicity, and efficiency, formats should be designed with more than one implementation style in mind. (In practice, pure stream I/O is adequate although random access makes it easier to write files.) It ought to be possible to read one of many objects in a file without scanning all the preceding data. Some programs need to read and play out their data in real time, so we need good compromises between generality and efficiency.

As much as we need standards, they can't hold up product schedules. So we also need a kind of decentralized extensibility where any software developer can define and refine new object types without some "standards authority" in the loop. Developers must be able to extend existing formats in a forward- and backward-compatible way. A central repository for design information and example programs can help us take full advantage of the standard.

For convenience, data formats should heed the restrictions of various processors and environments. E.g. word-alignment greatly helps 68000 access at insignificant cost to 8088 programs.

Other goals include the ability to share common elements over a list of objects and the ability to construct composite objects containing other data objects with structural information like directories.

And finally, "Simple things should be simple and complex things should be possible." Alan Kay.

Think Ahead

Let's think ahead and build programs that read and write files for each other and for programs yet to be designed. Build data formats to last for future computers so long as the overhead is acceptable. This extends the usefulness and life of today's programs and data.

To maximize interconnectivity, the standard file structure and the specific object formats must all be general and extensible. Think ahead when designing an object. It should serve many purposes and allow many programs to store and read back all the information they need; even squeeze in custom data. Then a programmer can store the available data and is encouraged to include fixed contextual details. Recipient programs can read the needed parts, skip unrecognized stuff, default missing data, and use the stored context to help transform the data as needed.

Scope

IFF addresses these needs by defining a standard file structure, some

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0869
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

initial data object types, ways to define new types, and rules for accessing these files. We can accomplish a great deal by writing programs according to this standard, but don't expect direct compatibility with existing software. We'll need conversion programs to bridge the gap from the old world.

IFF is geared for computers that readily process information in 8-bit bytes. It assumes a "physical layer" of data storage and transmission that reliably maintains "files" as strings of 8-bit bytes. The standard treats a "file" as a container of data bytes and is independent of how to find a file and whether it has a byte count.

This standard does not by itself implement a clipboard for cutting and pasting data between programs. A clipboard needs software to mediate access, to maintain a "contents version number" so programs can detect updates, and to manage the data in "virtual memory".

Data Abstraction

The basic problem is how to represent information in a way that's program-independent, compiler- independent, machine-independent, and device-independent.

The computer science approach is "data abstraction", also known as "objects", "actors", and "abstract data types". A data abstraction has a "concrete representation" (its storage format), an "abstract representation" (its capabilities and uses), and access procedures that isolate all the calling software from the concrete representation. Only the access procedures touch the data storage. Hiding mutable details behind an interface is called "information hiding". What data abstraction does is abstract from details of implementing the object, namely the selected storage representation and algorithms for manipulating it.

The power of this approach is modularity. By adjusting the access procedures we can extend and restructure the data without impacting the interface or its callers. Conversely, we can extend and restructure the interface and callers without making existing data obsolete. It's great for interchange!

But we seem to need the opposite: fixed file formats for all programs to access. Actually, we could file data abstractions ("filed objects") by storing the data and access procedures together. We'd have to encode the access procedures in a standard machine-independent programming language la PostScript. Even still, the interface can't evolve freely since we can't update all copies of the access procedures. So we'll have to design our abstract representations for limited evolution and occasional revolution (conversion).

In any case, today's microcomputers can't practically store data abstractions. They can do the next best thing: store arbitrary types of data in "data chunks", each with a type identifier and a length count. The type identifier is a reference by name to the access procedures (any local implementation). The length count enables storage-level object operations like "copy" and "skip to next" independent of object type.

Chunk writing is straightforward. Chunk reading requires a trivial parser to scan each chunk and dispatch to the proper access/conversion procedure. Reading chunks nested inside other chunks requires recursion, but no lookahead or backup.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0870
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

That's the main idea of IFF. There are, of course, a few other detailsI

Previous Work

Where our needs are similar, we borrow from existing standards.

Our basic need to move data between independently developed programs is similar to that addressed by the Apple Macintosh desk scrap or "clipboard" [Inside Macintosh chapter "Scrap Manager"]. The Scrap Manager works closely with the Resource Manager, a handy filer and swapper for data objects (text strings, dialog window templates, pictures, fontsI) including types yet to be designed [Inside Macintosh chapter "Resource Manager"]. The Resource Manager is a kin to Smalltalk's object swapper.

We will probably write a Macintosh desk accessory that converts IFF files to and from the Macintosh clipboard for quick and easy interchange with programs like MacPaint and Resource Mover.

Macintosh uses a simple and elegant scheme of 4-character "identifiers" to identify resource types, clipboard format types, file types, and file creator programs. Alternatives are unique ID numbers assigned by a central authority or by hierarchical authorities, unique ID numbers generated by algorithm, other fixed length character strings, and variable length strings. Character string identifiers double as readable signposts in data files and programs. The choice of 4 characters is a good tradeoff between storage space, fetch/compare/store time, and name space size. We'll honor Apple's designers by adopting this scheme.

"PICT" is a good example of a standard structured graphics format (including raster images) and its many uses [Inside Macintosh chapter "QuickDraw"]. Macintosh provides QuickDraw routines in ROM to create, manipulate, and display PICTs. Any application can create a PICT by simply asking QuickDraw to record a sequence of drawing commands. Since it's just as easy to ask QuickDraw to render a PICT to a screen or a printer, it's very effective to pass them between programs, say from an illustrator to a word processor. An important feature is the ability to store "comments" in a PICT which QuickDraw will ignore. Actually, it passes them to your optional custom "comment handler".

PostScript, Adobe's print file standard, is a more general way to represent any print image (which is a specification for putting marks on paper) [PostScript Language Manual]. In fact, PostScript is a full-fledged programming language. To interpret a PostScript program is to render a document on a raster output device. The language is defined in layers: a lexical layer of identifiers, constants, and operators; a layer of reverse polish semantics including scope rules and a way to define new subroutines; and a printing-specific layer of built-in identifiers and operators for rendering graphic images. It is clearly a powerful (Turing equivalent) image definition language. PICT and a subset of PostScript are candidates for structured graphics standards.

A PostScript document can be printed on any raster output device (including a display) but cannot generally be edited. That's because the original flexibility and constraints have been discarded. Besides, a PostScript program may use arbitrary computation to supply parameters like placement and size to each operator. A QuickDraw PICT, in comparison, is a more restricted format of graphic primitives parameterized by constants. So a PICT can be edited at the level of the primitives, e.g. move

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0871
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

or thicken a line. It cannot be edited at the higher level of, say, the bar chart data which generated the picture.

PostScript has another limitation: Not all kinds of data amount to marks on paper. A musical instrument description is one example. PostScript is just not geared for such uses.

"DIF" is another example of data being stored in a general format usable by future programs [DIF Technical Specification]. DIF is a format for spreadsheet data interchange. DIF and PostScript are both expressed in plain ASCII text files. This is very handy for printing, debugging, experimenting, and transmitting across modems. It can have substantial cost in compaction and read/write work, depending on use. We won't store IFF files this way but we could define an ASCII alternate representation with a converter program.

InterScript is Xerox' standard for interchange of editable documents [Introduction to InterScript]. It approaches a harder problem: How to represent editable word processor documents that may contain formatted text, pictures, cross-references like figure numbers, and even highly specialized objects like mathematical equations? InterScript aims to define one standard representation for each kind of information. Each InterScript-compatible editor is supposed to preserve the objects it doesn't understand and even maintain nested cross-references. So a simple word processor would let you edit the text of a fancy document without discarding the equations or disrupting the equation numbers.

Our task is similarly to store high level information and preserve as much content as practical while moving it between programs. But we need to span a larger universe of data types and cannot expect to centrally define them all. Fortunately, we don't need to make programs preserve information that they don't understand. And for better or worse, we don't have to tackle general-purpose cross-references yet.

□

3. Primitive Data Types

Atomic components such as integers and characters that are interpretable directly by the CPU are specified in one format for all processors. We chose a format that's most convenient for the Motorola MC68000 processor [M68000 16/32-Bit Microprocessor Programmer's Reference Manual].

N.B.: Part 3 dictates the format for "primitive" data types where and only where used in the overall file structure and standard kinds of chunks (Cf. Chunks). The number of such occurrences will be small enough that the costs of conversion, storage, and management of processor-specific files would far exceed the costs of conversion during I/O by "foreign" programs. A particular data chunk may be specified with a different format for its internal primitive types or with processor- or environment-speci fic variants if necessary to optimize local usage. Since that hurts data interchange, it's not recommended. (Cf. Designing New Data Sections, in Part 4.)

Alignment

All data objects larger than a byte are aligned on even byte addresses relative to the start of the file. This may require padding. Pad bytes are to be written as zeros, but don't count on that when reading.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0872
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

This means that every odd-length "chunk" (see below) must be padded so that the next one will fall on an even boundary. Also, designers of structures to be stored in chunks should include pad fields where needed to align every field larger than a byte. Zeros should be stored in all the pad bytes.

Justification: Even-alignment causes a little extra work for files that are used only on certain processors but allows 68000 programs to construct and scan the data in memory and do block I/O. You just add an occasional pad field to data structures that you're going to block read/write or else stream read/write an extra byte. And the same source code works on all processors. Unspecified alignment, on the other hand, would force 68000 programs to (dis)assemble word and long-word data one byte at a time. Pretty cumbersome in a high level language. And if you don't conditionally compile that out for other processors, you won't gain anything.

Numbers

Numeric types supported are two's complement binary integers in the format used by the MC68000 processor high byte first, high word first the reverse of 8088 and 6502 format. They could potentially include signed and unsigned 8, 16, and 32 bit integers but the standard only uses the following:

```
UBYTE    8 bits unsigned
WORD    16 bits signed
UWORD   16 bits unsigned
LONG    32 bits signed
```

The actual type definitions depend on the CPU and the compiler. In this document, we'll express data type definitions in the C programming language. [See C, A Reference Manual.] In 68000 Lattice C:

```
typedef unsigned char   UBYTE;  /*  8 bits unsigned     */
typedef short    WORD;  /* 16 bits signed       */
typedef unsigned short  UWORD;  /* 16 bits unsigned     */
typedef long     LONG;  /* 32 bits signed       */
```

Characters

The following character set is assumed wherever characters are used, e.g. in text strings, IDs, and TEXT chunks (see below).

Characters are encoded in 8-bit ASCII. Characters in the range NUL (hex 0) through DEL (hex 7F) are well defined by the 7-bit ASCII standard. IFF uses the graphic group RJS (SP, hex 20) through R~S (hex 7E).

Most of the control character group hex 01 through hex 1F have no standard meaning in IFF. The control character LF (hex 0A) is defined as a "newline" character. It denotes an intentional line break, that is, a paragraph or line terminator. (There is no way to store an automatic line break. That is strictly a function of the margins in the environment the text is placed.) The control character ESC (hex 1B) is a reserved escape character under the rules of ANSI standard 3.64-1979 American National Standard Additional Control Codes for Use with ASCII, ISO standard 2022, and ISO/DIS standard 6429.2.

Characters in the range hex 7F through hex FF are not globally defined

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0873
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

in IFF. They are best left reserved for future standardization. But note that the FORM type FTXT (formatted text) defines the meaning of these characters within FTXT forms. In particular, character values hex 7F through hex 9F are control codes while characters hex A0 through hex FF are extended graphic characters like , as per the ISO and ANSI standards cited above. [See the supplementary document "FTXT" IFF Formatted Text.]

Dates

A "creation date" is defined as the date and time a stream of data bytes was created. (Some systems call this a "last modified date".) Editing some data changes its creation date. Moving the data between volumes or machines does not.

The IFF standard date format will be one of those used in MS-DOS, Macintosh, or Amiga DOS (probably a 32-bit unsigned number of seconds since a reference point). Issue: Investigate these three.

Type IDs

A "type ID", "property name", "FORM type", or any other IFF identifier is a 32-bit value: the concatenation of four ASCII characters in the range R S (SP, hex 20) through R~S (hex 7E). Spaces (hex 20) should not precede printing characters; trailing spaces are ok. Control characters are forbidden.

```
typedef CHAR ID[4];
```

IDs are compared using a simple 32-bit case-dependent equality test.

Data section type IDs (aka FORM types) are restriced IDs. (Cf. Data Sections.) Since they may be stored in filename extensions (Cf. Single Purpose Files) lower case letters and punctuation marks are forbidden. Trailing spaces are ok.

Carefully choose those four characters when you pick a new ID. Make them mnemonic so programmers can look at an interchange format file and figure out what kind of data it contains. The name space makes it possible for developers scattered around the globe to generate ID values with minimal collisions so long as they choose specific names like "MUS4" instead of general ones like "TYPE" and "FILE". EA will "register" new FORM type IDs and format descriptions as they're devised, but collisions will be improbable so there will be no pressure on this "clearinghouse" process. Appendix A has a list of currently defined IDs.

Sometimes it's necessary to make data format changes that aren't backward compatible. Since IDs are used to denote data formats in IFF, new IDs are chosen to denote revised formats. Since programs won't read chunks whose IDs they don't recognize (see Chunks, below), the new IDs keep old programs from stumbling over new data. The conventional way to chose a "revision" ID is to increment the last character if it's a digit or else change the last character to a digit. E.g. first and second revisions of the ID "XY" would be "XY1" and "XY2". Revisions of "CMAP" would be "CMA1" and "CMA2".

Chunks

Chunks are the building blocks in the IFF structure. The form expressed

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0874
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

as a C typedef is:

```
typedef struct {
        ID      ckID;
        LONG    ckSize; /* sizeof(ckData) */
        UBYTE   ckData[/* ckSize */];
        } Chunk;
```

We can diagram an example chunk a "CMAP" chunk containing 12 data bytes like this:

```
                 ----------------
ckID:    |     'CMAP'     |
ckSize:  |       12       |
ckData:  | 0, 0, 0, 32    |    --------
         | 0, 0, 64, 0    |     12 bytes
         | 0, 0, 64, 0    |    ----------
         ----------------
```

The fixed header part means "Here's a type ckID chunk with ckSize bytes of data."

The ckID identifies the format and purpose of the chunk. As a rule, a program must recognize ckID to interpret ckData. It should skip over all unrecognized chunks. The ckID also serves as a format version number as long as we pick new IDs to identify new formats of ckData (see above).

The following ckIDs are universally reserved to identify chunks with particular IFF meanings: "LIST", "FORM", "PROP", "CAT ", and " ". The special ID "   " (4 spaces) is a ckID for "filler" chunks, that is, chunks that fill space but have no meaningful contents. The IDs "LIS1" through "LIS9", "FOR1" through "FOR9", and "CAT1" through "CAT9" are reserved for future "version number" variations. All IFF-compatible software must account for these 23 chunk IDs. Appendix A has a list of predefined IDs.

The ckSize is a logical block size how many data bytes are in ckData. If ckData is an odd number of bytes long, a 0 pad byte follows which is not included in ckSize. (Cf. Alignment.) A chunk's total physical size is ckSize rounded up to an even number plus the size of the header. So the smallest chunk is 8 bytes long with ckSize = 0. For the sake of following chunks, programs must respect every chunk's ckSize as a virtual end-of-file for reading its ckData even if that data is malformed, e.g. if nested contents are truncated.

We can describe the syntax of a chunk as a regular expression with "#" representing the ckSize, i.e. the length of the following {braced} bytes. The "[0]" represents a sometimes needed pad byte. (The regular expressions in this document are collected in Appendix A along with an explanation of notation.)

```
Chunk   ::= ID #{ UBYTE* } [0]
```

One chunk output technique is to stream write a chunk header, stream write the chunk contents, then random access back to the header to fill in the size. Another technique is to make a preliminary pass over the data to compute the size, then write it out all at once.

Strings, String Chunks, and String Properties

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0875
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

In a string of ASCII text, LF denotes a forced line break (paragraph or line terminator). Other control characters are not used. (Cf. Characters.)

The ckID for a chunk that contains a string of plain, unformatted text is "TEXT". As a practical matter, a text string should probably not be longer than 32767 bytes. The standard allows up to 231 - 1 bytes.

When used as a data property (see below), a text string chunk may be 0 to 255 characters long. Such a string is readily converted to a C string or a Pascal STRING[255]. The ckID of a property must be the property name, not "TEXT".

When used as a part of a chunk or data property, restricted C string format is normally used. That means 0 to 255 characters followed by a NUL byte (ASCII value 0).

Data Properties

Data properties specify attributes for following (non-property) chunks. A data property essentially says "identifier = value", for example "XY = (10, 200)", telling something about following chunks. Properties may only appear inside data sections ("FORM" chunks, cf. Data Sections) and property sections ("PROP" chunks, cf. Group PROP).

The form of a data property is a special case of Chunk. The ckID is a property name as well as a property type. The ckSize should be small since data properties are intended to be accumulated in RAM when reading a file. (256 bytes is a reasonable upper bound.) Syntactically:

Property ::= Chunk

When designing a data object, use properties to describe context information like the size of an image, even if they don't vary in your program. Other programs will need this information.

Think of property settings as assignments to variables in a programming language. Multiple assignments are redundant and local assignments temporarily override global assignments. The order of assignments doesn't matter as long as they precede the affected chunks. (Cf. LISTs, CATs, and Shared Properties.)

Each object type (FORM type) is a local name space for property IDs. Think of a "CMAP" property in a "FORM ILBM" as the qualified ID "ILBM.CMAP". Property IDs specified when an object type is designed (and therefore known to all clients) are called "standard" while specialized ones added later are "nonstandard".

Links

Issue: A standard mechanism for "links" or "cross references" is very desirable for things like combining images and sounds into animations. Perhaps we'll define "link" chunks within FORMs that refer to other FORMs or to specific chunks within the same and other FORMs. This needs further work. EA IFF 1985 has no standard link mechanism.

For now, it may suffice to read a list of, say, musical instruments, and then just refer to them within a musical score by index number.

File References

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0876
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Issue: We may need a standard form for references to other files. A "file ref" could name a directory and a file in the same type of operating system as the ref's originator. Following the reference would expect the file to be on some mounted volume. In a network environment, a file ref could name a server, too.

Issue: How can we express operating-system independent file refs?

Issue: What about a means to reference a portion of another file? Would this be a "file ref" plus a reference to a "link" within the target file?

□

4. Data Sections

The first thing we need of a file is to check: Does it contain IFF data and, if so, does it contain the kind of data we're looking for? So we come to the notion of a "data section".

A "data section" or IFF "FORM" is one self-contained "data object" that might be stored in a file by itself. It is one high level data object such as a picture or a sound effect. The IFF structure "FORM" makes it self- identifying. It could be a composite object like a musical score with nested musical instrument descriptions.

Group FORM

A data section is a chunk with ckID "FORM" and this arrangement:

```
FORM     ::= "FORM" #{ FormType (LocalChunk | FORM | LIST | CAT)*
}
FormType         ::= ID
LocalChunk       ::= Property | Chunk
```

The ID "FORM" is a syntactic keyword like "struct" in C. Think of a "struct ILBM" containing a field "CMAP". If you see "FORM" you'll know to expect a FORM type ID (the structure name, "ILBM" in this example) and a particular contents arrangement or "syntax" (local chunks, FORMs, LISTs, and CATs). (LISTs and CATs are discussed in part 5, below.) A "FORM ILBM", in particular, might contain a local chunk "CMAP", an "ILBM.CMAP" (to use a qualified name).

So the chunk ID "FORM" indicates a data section. It implies that the chunk contains an ID and some number of nested chunks. In reading a FORM, like any other chunk, programs must respect its ckSize as a virtual end-of-file for reading its contents, even if they're truncated.

The FormType (or FORM type) is a restricted ID that may not contain lower case letters or punctuation characters. (Cf. Type IDs. Cf. Single Purpose Files.)

The type-specific information in a FORM is composed of its "local chunks": data properties and other chunks. Each FORM type is a local name space for local chunk IDs. So "CMAP" local chunks in other FORM types may be unrelated to "ILBM.CMAP". More than that, each FORM type defines semantic scope. If you know what a FORM ILBM is, you'll know what an ILBM.CMAP is.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0877
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Local chunks defined when the FORM type is designed (and therefore known to all clients of this type) are called "standard" while specialized ones added later are "nonstandard".

Among the local chunks, property chunks give settings for various details like text font while the other chunks supply the essential information. This distinction is not clear cut. A property setting cancelled by a later setting of the same property has effect only on data chunks in between. E.g. in the sequence:

```
prop1 = x  (propN = value)*  prop1 = y
```

where the propNs are not prop1, the setting prop1 = x has no effect.

The following universal chunk IDs are reserved inside any FORM: "LIST", "FORM", "PROP", "CAT ", "JJJJ", "LIS1" through "LIS9", "FOR1" through "FOR9", and "CAT1" through "CAT9". (Cf. Chunks. Cf. Group LIST. Cf. Group PROP.) For clarity, these universal chunk names may not be FORM type IDs, either.

Part 5, below, talks about grouping FORMs into LISTs and CATs. They let you group a bunch of FORMs but don't impose any particular meaning or constraints on the grouping. Read on.

Composite FORMs

A FORM chunk inside a FORM is a full-fledged data section. This means you can build a composite object like a multi-frame animation sequence from available picture FORMs and sound effect FORMs. You can insert additional chunks with information like frame rate and frame count.

Using composite FORMs, you leverage on existing programs that create and edit the component FORMs. Those editors may even look into your composite object to copy out its type of component, although it'll be the rare program that's fancy enough to do that. Such editors are not allowed to replace their component objects within your composite object. That's because the IFF standard lets you specify consistency requirements for the composite FORM such as maintaining a count or a directory of the components. Only programs that are written to uphold the rules of your FORM type should create or modify such FORMs.

Therefore, in designing a program that creates composite objects, you are strongly requested to provide a facility for your users to import and export the nested FORMs. Import and export could move the data through a clipboard or a file.

Here are several existing FORM types and rules for defining new ones.

FTXT

An FTXT data section contains text with character formatting information like fonts and faces. It has no paragraph or document formatting information like margins and page headers. FORM FTXT is well matched to the text representation in Amiga's Intuition environment. See the supplemental document "FTXT" IFF Formatted Text.

ILBM

"ILBM" is an InterLeaved BitMap image with color map; a machine-independent format for raster images. FORM ILBM is the standard image file format

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0878
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

for the Commodore-Amiga computer and is useful in other environments, too. See the supplemental document "ILBM" IFF Interleaved Bitmap.

PICS

The data chunk inside a "PICS" data section has ID "PICT" and holds a QuickDraw picture. Issue: Allow more than one PICT in a PICS? See Inside Macintosh chapter "QuickDraw" for details on PICTs and how to create and display them on the Macintosh computer.

The only standard property for PICS is "XY", an optional property that indicates the position of the PICT relative to "the big picture". The contents of an XY is a QuickDraw Point.

Note: PICT may be limited to Macintosh use, in which case there'll be another format for structured graphics in other environments.

Other Macintosh Resource Types

Some other Macintosh resource types could be adopted for use within IFF files; perhaps MWRT, ICN, ICN#, and STR#.

Issue: Consider the candidates and reserve some more IDs.

Designing New Data Sections

Supplemental documents will define additional object types. A supplement needs to specify the object's purpose, its FORM type ID, the IDs and formats of standard local chunks, and rules for generating and interpreting the data. It's a good idea to supply typedefs and an example source program that accesses the new object. See "ILBM" IFF Interleaved Bitmap for a good example.

Anyone can pick a new FORM type ID but should reserve it with Electronic Arts at their earliest convenience. [Issue: EA contact person? Hand this off to another organization?] While decentralized format definitions and extensions are possible in IFF, our preference is to get design consensus by committee, implement a program to read and write it, perhaps tune the format, and then publish the format with example code. Some organization should remain in charge of answering questions and coordinating extensions to the format.

If it becomes necessary to revise the design of some data section, its FORM type ID will serve as a version number (Cf. Type IDs). E.g. a revised "VDEO" data section could be called "VDE1". But try to get by with compatible revisions within the existing FORM type.

In a new FORM type, the rules for primitive data types and word-alignment (Cf. Primitive Data Types) may be overriden for the contents of its local chunks but not for the chunk structure itself if your documentation spells out the deviations. If machine-specific type variants are needed, e.g. to store vast numbers of integers in reverse bit order, then outline the conversion algorithm and indicate the variant inside each file, perhaps via different FORM types. Needless to say, variations should be minimized.

In designing a FORM type, encapsulate all the data that other programs will need to interpret your files. E.g. a raster graphics image should specify the image size even if your program always uses 320 x 200 pixels x 3 bitplanes. Receiving programs are then empowered to append

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0879
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

or clip the image rectangle, to add or drop bitplanes, etc. This enables a lot more compatibility.

Separate the central data (like musical notes) from more specialized information (like note beams) so simpler programs can extract the central parts during read-in. Leave room for expansion so other programs can squeeze in new kinds of information (like lyrics). And remember to keep the property chunks manageably short let's say 2 256 bytes.

When designing a data object, try to strike a good tradeoff between a super-general format and a highly-specialized one. Fit the details to at least one particular need, for example a raster image might as well store pixels in the current machine's scan order. But add the kind of generality that makes it usable with foreseeable hardware and software. E.g. use a whole byte for each red, green, and blue color value even if this year's computer has only 4-bit video DACs. Think ahead and help other programs so long as the overhead is acceptable. E.g. run compress a raster by scan line rather than as a unit so future programs can swap images by scan line to and from secondary storage.

Try to design a general-purpose "least common multiple" format that encompasses the needs of many programs without getting too complicated. Let's coalesce our uses around a few such formats widely separated in the vast design space. Two factors make this flexibility and simplicity practical. First, file storage space is getting very plentiful, so compaction is not a priority. Second, nearly any locally-performed data conversion work during file reading and writing will be cheap compared to the I/O time.

It must be ok to copy a LIST or FORM or CAT intact, e.g. to incorporate it into a composite FORM. So any kind of internal references within a FORM must be relative references. They could be relative to the start of the containing FORM, relative from the referencing chunk, or a sequence number into a collection.

With composite FORMs, you leverage on existing programs that create and edit the components. If you write a program that creates composite objects, please provide a facility for your users to import and export the nested FORMs. The import and export functions may move data through a separate file or a clipboard.

Finally, don't forget to specify all implied rules in detail.

□

5. LISTs, CATs, and Shared Properties

Data often needs to be grouped together like a list of icons. Sometimes a trick like arranging little images into a big raster works, but generally they'll need to be structured as a first class group. The objects "LIST" and "CAT" are IFF-universal mechanisms for this purpose.

Property settings sometimes need to be shared over a list of similar objects. E.g. a list of icons may share one color map. LIST provides a means called "PROP" to do this. One purpose of a LIST is to define the scope of a PROP. A "CAT", on the other hand, is simply a concatenation of objects.

Simpler programs may skip LISTs and PROPs altogether and just handle FORMs and CATs. All "fully-conforming" IFF programs also know about

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0880
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

"CAT ", "LIST", and "PROP". Any program that reads a FORM inside a LIST must process shared PROPs to correctly interpret that FORM.

Group CAT

A CAT is just an untyped group of data objects.

Structurally, a CAT is a chunk with chunk ID "CAT " containing a "contents type" ID followed by the nested objects. The ckSize of each contained chunk is essentially a relative pointer to the next one.

```
CAT     ::= "CAT " #{ ContentsType (FORM | LIST | CAT)* }
ContentsType    ::= ID  -- a hint or an "abstract data type" ID
```

In reading a CAT, like any other chunk, programs must respect it's ckSize as a virtual end-of-file for reading the nested objects even if they're malformed or truncated.

The "contents type" following the CAT's ckSize indicates what kind of FORMs are inside. So a CAT of ILBMs would store "ILBM" there. It's just a hint. It may be used to store an "abstract data type". A CAT could just have blank contents ID ("JJJJ") if it contains more than one kind of FORM.

CAT defines only the format of the group. The group's meaning is open to interpretation. This is like a list in LISP: the structure of cells is predefined but the meaning of the contents as, say, an association list depends on use. If you need a group with an enforced meaning (an "abstract data type" or Smalltalk "subclass"), some consistency constraints, or additional data chunks, use a composite FORM instead (Cf. Composite FORMs).

Since a CAT just means a concatenation of objects, CATs are rarely nested. Programs should really merge CATs rather than nest them.

Group LIST

A LIST defines a group very much like CAT but it also gives a scope for PROPs (see below). And unlike CATs, LISTs should not be merged without understanding their contents.

Structurally, a LIST is a chunk with ckID "LIST" containing a "contents type" ID, optional shared properties, and the nested contents (FORMs, LISTs, and CATs), in that order. The ckSize of each contained chunk is a relative pointer to the next one. A LIST is not an arbitrary linked list the cells are simply concatenated.

```
LIST    ::= "LIST" #{ ContentsType PROP* (FORM | LIST | CAT)* }
ContentsType    ::= ID
```

Group PROP

PROP chunks may appear in LISTs (not in FORMs or CATs). They supply shared properties for the FORMs in that LIST. This ability to elevate some property settings to shared status for a list of forms is useful for both indirection and compaction. E.g. a list of images with the same size and colors can share one "size" property and one "color map" property. Individual FORMs can override the shared settings.

The contents of a PROP is like a FORM with no data chunks:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0881
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
PROP    ::= "PROP" #{ FormType Property* }

It means, "Here are the shared properties for FORM type <<FormType>."

A LIST may have at most one PROP of a FORM type, and all the PROPs
must appear before any of the FORMs or nested LISTs and CATs. You
can have subsequences of FORMs sharing properties by making each subsequence
a LIST.

Scoping: Think of property settings as variable bindings in nested
blocks of a programming language. Where in C you could write:

TEXT_FONT text_font = Courier;  /* program's global default     */

File(); {
        TEXT_FONT text_font = TimesRoman;       /* shared setting       */

                {
                TEXT_FONT text_font = Helvetica;  /* local setting      */
                Print("Hello ");          /* uses font Helvetica */
                }

                {
                Print("there.");          /* uses font TimesRoman */
                }
        }

An IFF file could contain:

LIST {
        PROP TEXT {
                FONT {TimesRoman}       /* shared setting       */
                }

        FORM TEXT {
                FONT {Helvetica}        /* local setting        */
                CHRS {Hello }           /* uses font Helvetica  */
                }

        FORM TEXT {
                CHRS {there.}   /* uses font TimesRoman */
                }
        }

The shared property assignments selectively override the reader's
global defaults, but only for FORMs within the group. A FORM's own
property assignments selectively override the global and group-supplied
values. So when reading an IFF file, keep property settings on a stack.
They're designed to be small enough to hold in main memory.

Shared properties are semantically equivalent to copying those properties
into each of the nested FORMs right after their FORM type IDs.

Properties for LIST

Optional "properties for LIST" store the origin of the list's contents
in a PROP chunk for the fake FORM type "LIST". They are the properties
originating program "OPGM", processor family "OCPU", computer type
"OCMP", computer serial number or network address "OSN ", and user
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0882
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

name "UNAM". In our imperfect world, these could be called upon to distinguish between unintended variations of a data format or to work around bugs in particular originating/receiving program pairs. Issue: Specify the format of these properties.

A creation date could also be stored in a property but let's ask that file creating, editing, and transporting programs maintain the correct date in the local file system. Programs that move files between machine types are expected to copy across the creation dates.

□

6. Standard File Structure

File Structure Overview

An IFF file is just a single chunk of type FORM, LIST, or CAT. Therefore an IFF file can be recognized by its first 4 bytes: "FORM", "LIST", or "CAT ". Any file contents after the chunk's end are to be ignored.

Since an IFF file can be a group of objects, programs that read/write single objects can communicate to an extent with programs that read/write groups. You're encouraged to write programs that handle all the objects in a LIST or CAT. A graphics editor, for example, could process a list of pictures as a multiple page document, one page at a time.

Programs should enforce IFF's syntactic rules when reading and writing files. This ensures robust data transfer. The public domain IFF reader/writer subroutine package does this for you. A utility program "IFFCheck" is available that scans an IFF file and checks it for conformance to IFF's syntactic rules. IFFCheck also prints an outline of the chunks in the file, showing the ckID and ckSize of each. This is quite handy when building IFF programs. Example programs are also available to show details of reading and writing IFF files.

A merge program "IFFJoin" will be available that logically appends IFF files into a single CAT group. It "unwraps" each input file that is a CAT so that the combined file isn't nested CATs.

If we need to revise the IFF standard, the three anchoring IDs will be used as "version numbers". That's why IDs "FOR1" through "FOR9", "LIS1" through "LIS9", and "CAT1" through "CAT9" are reserved.

IFF formats are designed for reasonable performance with floppy disks. We achieve considerable simplicity in the formats and programs by relying on the host file system rather than defining universal grouping structures like directories for LIST contents. On huge storage systems, IFF files could be leaf nodes in a file structure like a B-tree. Let's hope the host file system implements that for us!

Thre are two kinds of IFF files: single purpose files and scrap files. They differ in the interpretation of multiple data objects and in the file's external type.

Single Purpose Files

A single purpose IFF file is for normal "document" and "archive" storage. This is in contrast with "scrap files" (see below) and temporary backing storage (non-interchange files).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0883
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The external file type (or filename extension, depending on the host file system) indicates the file's contents. It's generally the FORM type of the data contained, hence the restrictions on FORM type IDs.

Programmers and users may pick an "intended use" type as the filename extension to make it easy to filter for the relevant files in a filename requestor. This is actually a "subclass" or "subtype" that conveniently separates files of the same FORM type that have different uses. Programs cannot demand conformity to its expected subtypes without overly restricting data interchange since they cannot know about the subtypes to be used by future programs that users will want to exchange data with.

Issue: How to generate 3-letter MS-DOS extensions from 4-letter FORM type IDs?

Most single purpose files will be a single FORM (perhaps a composite FORM like a musical score containing nested FORMs like musical instrument descriptions). If it's a LIST or a CAT, programs should skip over unrecognized objects to read the recognized ones or the first recognized one. Then a program that can read a single purpose file can read something out of a "scrap file", too.

Scrap Files

A "scrap file" is for maximum interconnectivity in getting data between programs; the core of a clipboard function. Scrap files may have type "IFF " or filename extension ".IFF".

A scrap file is typically a CAT containing alternate representations of the same basic information. Include as many alternatives as you can readily generate. This redundancy improves interconnectivity in situations where we can't make all programs read and write super-general formats. [Inside Macintosh chapter "Scrap Manager".] E.g. a graphically-annotated musical score might be supplemented by a stripped down 4-voice melody and by a text (the lyrics).

The originating program should write the alternate representations in order of "preference": most preferred (most comprehensive) type to least preferred (least comprehensive) type. A receiving program should either use the first appearing type that it understands or search for its own "preferred" type.

A scrap file should have at most one alternative of any type. (A LIST of same type objects is ok as one of the alternatives.) But don't count on this when reading; ignore extra sections of a type. Then a program that reads scrap files can read something out of single purpose files.

Rules for Reader Programs

Here are some notes on building programs that read IFF files. If you use the standard IFF reader module "IFFR.C", many of these rules and details will be automatically handled. (See "Support Software" in Appendix A.) We recommend that you start from the example program "ShowILBM.C". You should also read up on recursive descent parsers. [See, for example, Compiler Construction, An Advanced Course.]

% The standard is very flexible so many programs can exchange data. This implies a program has to scan the file and react to what's actually there in whatever order it appears. An IFF reader program

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0884
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

is a parser.

% For interchange to really work, programs must be willing to do some conversion during read-in. If the data isn't exactly what you expect, say, the raster is smaller than those created by your program, then adjust it. Similarly, your program could crop a large picture, add or drop bitplanes, and create/discard a mask plane. The program should give up gracefully on data that it can't convert.

% If it doesn't start with "FORM", "LIST", or "CAT ", it's not an IFF-85 file.

% For any chunk you encounter, you must recognize its type ID to understand its contents.

% For any FORM chunk you encounter, you must recognize its FORM type ID to understand the contained "local chunks". Even if you don't recognize the FORM type, you can still scan it for nested FORMs, LISTs, and CATs of interest.

% Don't forget to skip the pad byte after every odd-length chunk.

% Chunk types LIST, FORM, PROP, and CAT are generic groups. They always contain a subtype ID followed by chunks.

% Readers ought to handle a CAT of FORMs in a file. You may treat the FORMs like document pages to sequence through or just use the first FORM.

% Simpler IFF readers completely skip LISTs. "Fully IFF-conforming" readers are those that handle LISTs, even if just to read the first FORM from a file. If you do look into a LIST, you must process shared properties (in PROP chunks) properly. The idea is to get the correct data or none at all.

% The nicest readers are willing to look into unrecognized FORMs for nested FORM types that they do recognize. For example, a musical score may contain nested instrument descriptions and an animation file may contain still pictures.

Note to programmers: Processing PROP chunks is not simple! You'll need some background in interpreters with stack frames. If this is foreign to you, build programs that read/write only one FORM per file. For the more intrepid programmers, the next paragraph summarizes how to process LISTs and PROPs. See the general IFF reader module "IFFR.C" and the example program "ShowILBM.C" for details.

Allocate a stack frame for every LIST and FORM you encounter and initialize it by copying the stack frame of the parent LIST or FORM. At the top level, you'll need a stack frame initialized to your program's global defaults. While reading each LIST or FORM, store all encountered properties into the current stack frame. In the example ShowILBM, each stack frame has a place for a bitmap header property ILBM.BMHD and a color map property ILBM.CMAP. When you finally get to the ILBM's BODY chunk, use the property settings accumulated in the current stack frame.

An alternate implementation would just remember PROPs encountered, forgetting each on reaching the end of its scope (the end of the containing LIST). When a FORM XXXX is encountered, scan the chunks in all remembered PROPs XXXX, in order, as if they appeared before the chunks actually

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0885
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

in the FORM XXXX. This gets trickier if you read FORMs inside of FORMs.

Rules for Writer Programs

Here are some notes on building programs that write IFF files, which is much easier than reading them. If you use the standard IFF writer module "IFFW.C" (see "Support Software" in Appendix A), many of these rules and details will automatically be enforced. See the example program "Raw2ILBM.C".

% An IFF file is a single FORM, LIST, or CAT chunk.

% Any IFF-85 file must start with the 4 characters "FORM", "LIST", or "CAT ", followed by a LONG ckSize. There should be no data after the chunk end.

% Chunk types LIST, FORM, PROP, and CAT are generic. They always contain a subtype ID followed by chunks. These three IDs are universally reserved, as are "LIS1" through "LIS9", "FOR1" through "FOR9", "CAT1" through "CAT9", and " ".

% Don't forget to write a 0 pad byte after each odd-length chunk.

% Four techniques for writing an IFF group: (1) build the data in a file mapped into virtual memory, (2) build the data in memory blocks and use block I/O, (3) stream write the data piecemeal and (don't forget!) random access back to set the group length count, and (4) make a preliminary pass to compute the length count then stream write the data.

% Do not try to edit a file that you don't know how to create. Programs may look into a file and copy out nested FORMs of types that they recognize, but don't edit and replace the nested FORMs and don't add or remove them. That could make the containing structure inconsistent. You may write a new file containing items you copied (or copied and modified) from another IFF file, but don't copy structural parts you don't understand.

% You must adhere to the syntax descriptions in Appendex A. E.g. PROPs may only appear inside LISTs.

□

Appendix A. Reference

Type Definitions

The following C typedefs describe standard IFF structures. Declarations to use in practice will vary with the CPU and compiler. For example, 68000 Lattice C produces efficient comparison code if we define ID as a "LONG". A macro "MakeID" builds these IDs at compile time.

```
/* Standard IFF types, expressed in 68000 Lattice C.    */

typedef unsigned char UBYTE;    /*  8 bits unsigned     */
typedef short WORD;     /* 16 bits signed       */
typedef unsigned short UWORD;   /* 16 bits unsigned     */
typedef long LONG;      /* 32 bits signed       */
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0886
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
typedef char ID[4];     /* 4 chars in ' ' through '~'   */

typedef struct {
        ID      ckID;
        LONG    ckSize; /* sizeof(ckData)       */
        UBYTE   ckData[/* ckSize */];
        } Chunk;

/* ID typedef and builder for 68000 Lattice C. */
typedef LONG ID;        /* 4 chars in ' ' through '~'   */
#define MakeID(a,b,c,d) ( (a)<<<<24 | (b)<<<<16 | (c)<<<<8 | (d) )

/* Globally reserved IDs. */
#define ID_FORM    MakeID('F','O','R','M')
#define ID_LIST    MakeID('L','I','S','T')
#define ID_PROP    MakeID('P','R','O','P')
#define ID_CAT     MakeID('C','A','T',' ')
#define ID_FILLER MakeID(' ',' ',' ',' ')
```

Syntax Definitions

Here's a collection of the syntax definitions in this document.

```
Chunk   ::= ID #{ UBYTE* } [0]

Property        ::= Chunk

FORM    ::= "FORM" #{ FormType (LocalChunk | FORM | LIST | CAT)*
}
FormType        ::= ID
LocalChunk      ::= Property | Chunk

CAT     ::= "CAT " #{ ContentsType (FORM | LIST | CAT)* }
ContentsType    ::= ID  -- a hint or an "abstract data type" ID

LIST    ::= "LIST" #{ ContentsType PROP* (FORM | LIST | CAT)* }
PROP    ::= "PROP" #{ FormType Property* }
```

In this extended regular expression notation, the token "#" represents a ckSize LONG count of the following {braced} data bytes. Literal items are shown in "quotes", [square bracketed items] are optional, and "*" means 0 or more instances. A sometimes-needed pad byte is shown as "[0]".

Defined Chunk IDs

This is a table of currently defined chunk IDs. We may also borrow some Macintosh IDs and data formats.

Group chunk IDs
        FORM, LIST, PROP, CAT.
Future revision group chunk IDs
        FOR1 I FOR9, LIS1 I LIS9, CAT1 I CAT9.
FORM type IDs
        (The above group chunk IDs may not be used for FORM type IDs.)
        (Lower case letters and punctuation marks are forbidden in FORM type IDs.)
        8SVX 8-bit sampled sound voice, ANBM animated bitmap, FNTR raster font, FNTV vector font, FTXT formatted text, GSCR general-use musical score, ILBM interleaved raster bitmap image, PDEF Deluxe Print page

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0887
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

definition, PICS Macintosh picture, PLBM (obsolete), USCR Uhuru Sound Software musical score, UVOX Uhuru Sound Software Macintosh voice, SMUS simple musical score, VDEO Deluxe Video Construction Set video.
Data chunk IDs
"JJJJ", TEXT, PICT.
PROP LIST property IDs
OPGM, OCPU, OCMP, OSN, UNAM.

□
Support Software

These public domain C source programs are available for use in building IFF-compatible programs:

IFF.H, IFFR.C, IFFW.C

IFF reader and writer package.
These modules handle many of the details of reliably reading and writing IFF files.

IFFCheck.C This handy utility program scans an IFF file, checks that the contents are well formed, and prints an outline of the chunks.

PACKER.H, Packer.C, UnPacker.C

Run encoder and decoder used for ILBM files.

ILBM.H, ILBMR.C, ILBMW.C

Reader and writer support routines for raster image FORM ILBM. ILBMR calls IFFR and UnPacker. ILBMW calls IFFW and Packer.

ShowILBM.C
Example caller of IFFR and ILBMR modules. This Commodore-Amiga program reads and displays a FORM ILBM.
Raw2ILBM.C
Example ILBM writer program. As a demonstration, it reads a raw raster image file and writes the image as a FORM ILBM file.
ILBM2Raw.C
Example ILBM reader program. Reads a FORM ILBM file and writes it into a raw raster image.

REMALLOC.H, Remalloc.c

Memory allocation routines used in these examples.

INTUALL.H generic "include almost everything" include-file with the sequence of includes correctly specified.

READPICT.H, ReadPict.c

given an ILBM file, read it into a bitmap and a color map

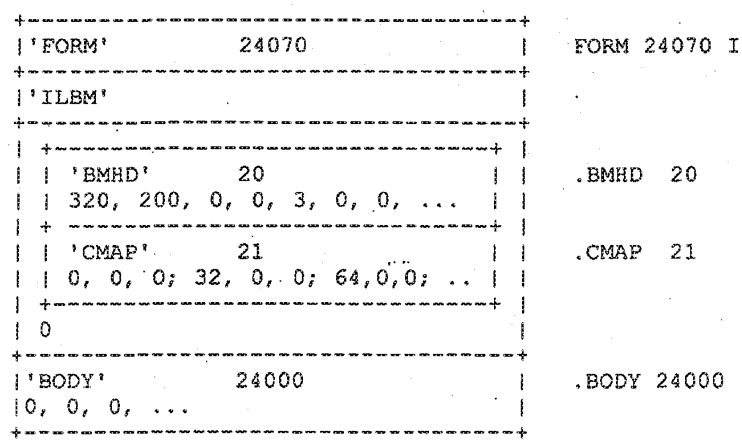
PUTPICT.H, PutPict.c

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0888
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

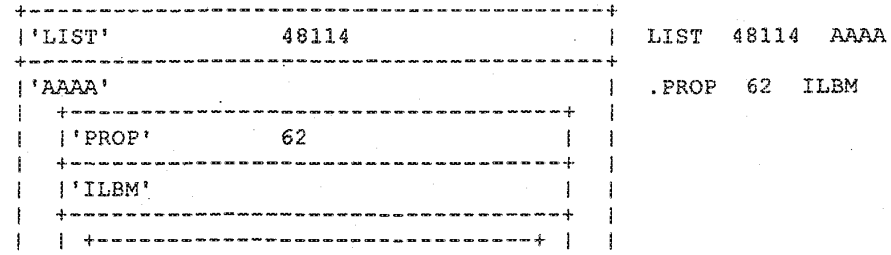
	given a bitmap and a color map, save it as an ILBM file.
GIO.H, Gio.c	generic I/O speedup package. Attempts to speed disk I/O by buffering writes and reads.
giocall.c	sample call to gio.
ilbmdump.c	reads in ILBM file, prints out ascii representation for including in C files.
bmprintc.c	prints out a C-language representation of data for a bitmap.

□
Example Diagrams

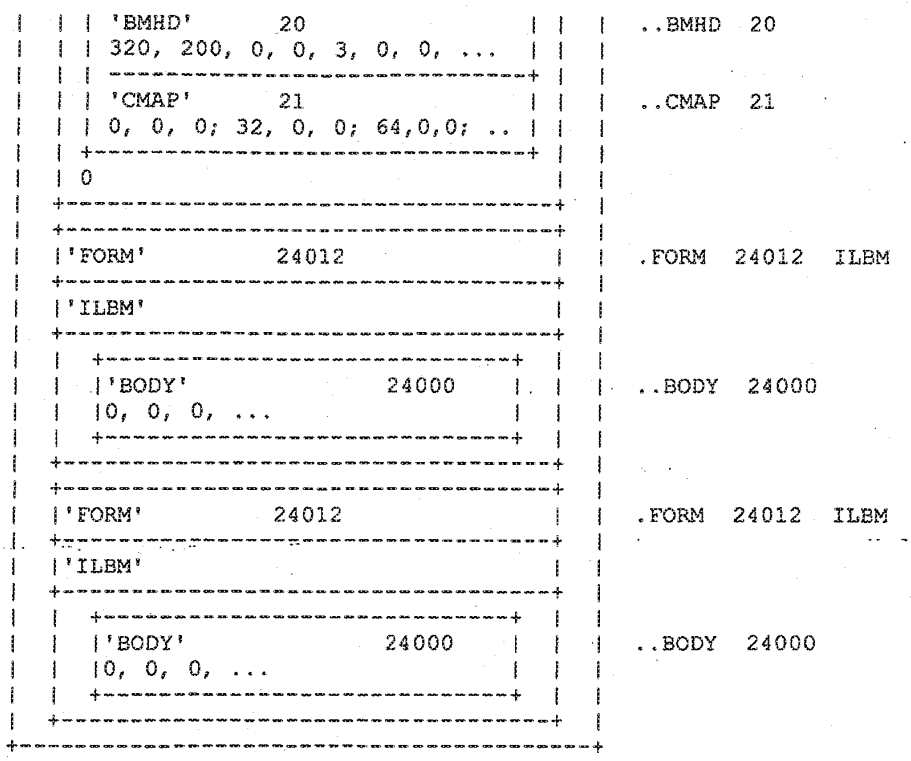
Here's a box diagram for an example IFF file, a raster image FORM ILBM. This FORM contains a bitmap header property chunk BMHD, a color map property chunk CMAP, and a raster data chunk BODY. This particular raster is 320 x 200 pixels x 3 bit planes uncompressed. The "0" after the CMAP chunk represents a zero pad byte; included since the CMAP chunk has an odd length. The text to the right of the diagram shows the outline that would be printed by the IFFCheck utility program for this particular file.



This second diagram shows a LIST of two FORMs ILBM sharing a common BMHD property and a common CMAP property. Again, the text on the right is an outline a la IFFCheck.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0889
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



□

Appendix B. Standards Committee

The following people contributed to the design of this IFF standard:

Bob "Kodiak" Burns, Commodore-Amiga
R. J. Mical, Commodore-Amiga
Jerry Morrison, Electronic Arts
Greg Riker, Electronic Arts
Steve Shaw, Electronic Arts
Barry Walsh, Commodore-Amiga

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0890
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Adaptive streaming

Expression Studio 2.0

Adaptive streaming is a process that adjusts the quality of a video delivered to a web page based on changing network conditions to ensure the best possible viewer experience. Adaptive streaming is made possible by changes introduced in Microsoft Silverlight 2.

Internet connection speeds vary widely, and the speed of each type of connection also varies depending on a wide variety of conditions. For example, if a user connects to an ISP at 56 Kbps, that does not mean that 56 Kbps of bandwidth is available at all times. Bandwidth can vary, meaning that a 56-Kbps connection may decrease or increase based on current network conditions, causing video quality to fluctuate as well. Adaptive streaming adjusts the bit rate of the video to adapt to changing network conditions.

Adaptive streaming simplifies content creation and management, making streaming video easy to deploy, and does not require any coding. For viewers who use Microsoft Windows Media Player rather than Silverlight 2, Windows Media Multiple Bit Rate encoding is available.

To use adaptive streaming, you need Microsoft Expression Encoder 2 with Service Pack 1, Microsoft Windows Server 2008, Microsoft Internet Information Services (IIS) 7.0, the IIS Smooth Streaming module, and Silverlight 2.

Adaptive streaming and Expression Encoder 2

When you encode video for adaptive streaming using Expression Encoder 2, the video is output in IIS Smooth Streaming format. The video files can then be published to IIS by using the WebDAV publishing plug-in available by default in the Publish section of the Output tab. Once the video is published, IIS then delivers it to the Silverlight client for viewing by the user.

To encode video for adaptive streaming

1. Import the video to be encoded by clicking File, locating the video to be encoded, and then clicking Import.
2. On the Encode panel, in the Video dropdown list in the Profile category, click Adaptive Streaming.

> **Note:**
> The Adaptive Streaming profile encodes the video at four different bit rates.

1. In the Container category, in the Output Mode dropdown, click IIS Smooth Streaming.
2. On the Output panel, in the Template dropdown list in the Job Output category, click a Silverlight 2 template.
3. Publish the video to a Windows Server that is running IIS and has the Smooth Streaming module installed. For more information, see Publish a video.

See also

Concepts

Encode your video for Microsoft Silverlight and the web
Publish a video

Community Content

Smooth Streaming Deployment Guide

Microsoft® Expression® Encoder

Windows Server® Internet Information Services

Microsoft® Silverlight®

*Author: **David M. Nelson***
Microsoft Corporation
Last Updated: August 2010

Table of Contents

Introduction 4

Encoding Smooth Streams 4

System Requirements 5

Encoding On-Demand Smooth Streams using Expression Encoder 5
Encoding media files 6

Publishing your media 12
Enabling WebDAV publishing on the Web server 12
Publishing the Smooth Streaming presentation to the Web server 14
Copying the Smooth Streaming presentation to the Web server 16

Encoding Live Smooth Streams Using Expression Encoder 18
Encoding a live broadcast 18

Serving Smooth Streams 23

System Requirements 23

Installing the Smooth Streaming Extensions 24

Serving On-Demand Smooth Streams 25
Serving an on-demand Smooth Streaming presentation 26
Viewing Smooth Streaming presentation details 26
Editing a Smooth Streaming presentation 29
Copying a Smooth Streaming presentation 31
Configuring advanced settings 33

Serving Live Smooth Streams 36
Creating a publishing point 37
About publishing point roles 40
Broadcasting live streams 43
Distributing live streams 46
Archiving live streams 48
Monitoring publishing points 49
State 49
Archive 50
Fragment 50
Stream 51
Managing credentials 51
Configuring credentials 51
Applying credentials 54
Configuring advanced settings 58
Configuring server cache settings 59
Configuring the archive path 60
Configuring client cache settings 61

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0894
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Playing Smooth Streams 62
Installing the sample Smooth Streaming player 62
Enabling playback of Smooth Streams 63
Additional Resources 65
Legal Notice 66

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0895
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Introduction

Smooth Streaming is the Microsoft implementation of adaptive streaming technology, which is a form of Web-based media content delivery that uses standard HTTP. Instead of delivering media as full-file downloads, or as persistent (and thus stateful) streams, the content is delivered to clients as a series of MPEG-4 (MP4) fragments that can be cached at edge servers. Smooth Streaming-compatible clients use special heuristics to dynamically monitor current network and local PC conditions and seamlessly switch the video quality of the Smooth Streaming presentation that they receive. As clients play the fragments, network conditions may change (for example, bandwidth may decrease) or video processing may be impacted by other applications that are running. Clients can immediately request that the next fragment come from a stream that is encoded at a different bit rate to accommodate the changing conditions. This enables clients to play the media without stuttering, buffering, or freezing. As a result, users experience the highest-quality playback available, with no interruptions in the stream.

Smooth Streaming provides media companies with a better way to make full-HD (1080p) streaming on the Web a reality. It offers code-free deployment and simplified content management for content creators and content delivery networks. For users, the improved viewing experience brings the reliability and quality of television to their favorite video Web sites.

This document describes the Microsoft implementation for delivering a full Smooth Streaming experience. As a content producer, you can encode on-demand and live Smooth Streaming video using Microsoft® Expression® Encoder. As a content provider, you can use IIS Media Services to serve the encoded Smooth Streams. And as a content consumer, you can play the Smooth Streams using a Smooth Streaming-compatible client, such as Microsoft® Silverlight®.

Note

- To see Smooth Streaming in action, go to the Experience Smooth Streaming Web page (http://go.microsoft.com/fwlink/?LinkId=165653), where you can use a Silverlight browser plug-in to simulate different bandwidths and see how Smooth Streaming responds.

Encoding Smooth Streams

Smooth Streaming encoders convert prerecorded or live audio and video content to the Microsoft implementation of the adaptive streaming format. You can encode prerecorded media files that are already stored on your hard disk or the live output from hardware recording devices such as digital video cameras, USB webcams, or SDI and PCI capture cards.

This section describes how to use Microsoft Expression Encoder 4 Pro (http://go.microsoft.com/fwlink/?LinkId=165787) to create on-demand Smooth Streaming files and live Smooth Streaming broadcasts.

There are also many third-party encoding solutions available to you for encoding on-demand and live Smooth Streaming video. To help you determine which encoding solution is best-suited for your Smooth

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0896
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Streaming configuration, see the list of our Encoding Partners (http://go.microsoft.com/fwlink/?LinkId=165785).

System Requirements

Software:

- Microsoft Expression Encoder 4 Pro (http://go.microsoft.com/fwlink/?LinkId=165787). For encoding on-demand Smooth Streaming files and live Smooth Streaming broadcasts.
- Microsoft Silverlight 4 (http://go.microsoft.com/fwlink/?LinkId=165789). For previewing your encoded Smooth Streaming presentation before you publish it to a Smooth Streaming-enabled server.
- Microsoft .NET Framework 4 (http://go.microsoft.com/fwlink/?LinkId=165793).
- Desktop Experience (http://go.microsoft.com/fwlink/?LinkId=199932) (server operating systems only).

Operating System: Windows Server® 2008 R2, Windows Server® 2008, Windows® 7, Windows Vista®, or Windows® XP SP3.

Processor: 1.0 GHz or faster (2.6 GHz or faster processor with 6 or more logical cores is recommended).

Memory: 1 GB RAM (4 GB recommended).

Hard Drive: 2 GB or more of available hard-disk space.

Graphics processor: Microsoft DirectX® 9.0 graphics with Windows Display Driver Model (WDDM) Driver, 128 MB of graphics RAM or more, Pixel Shader 3.0 in hardware, 32-bits per pixel.

Monitor: 1024 x 768 or higher resolution with 24-bit color.

Encoding On-Demand Smooth Streams using Expression Encoder

With a few mouse clicks in Expression Encoder 4 Pro, you can encode several quality levels of a specific piece of content, with each quality level encoded as its own complete file. You can then choose a Microsoft Silverlight (http://go.microsoft.com/fwlink/?LinkId=165789) player template to include with the encoded output and publish all of the files directly from Expression Encoder to a Smooth Streaming-enabled Web server.

When you encode media for Smooth Streaming in Expression Encoder, it creates a Smooth Streaming presentation that contains the following file types:

- ***.ismv**. MP4 container files that contain MP4 video fragments (and audio fragments if the video source also contains audio). Expression Encoder creates one .ismv file per bit rate, and the number of bit rates depends on the IIS Smooth Streaming preset that you select. For example, if you select a preset that specifies that the video be encoded using nine different bit rates, Expression Encoder creates nine .ismv files.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0897
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- ***.isma**. MP4 container files that contain only audio fragments. If you encode an audio-only source, this is the file format that results. As with .ismv files, the number of .isma files that are created can vary depending on your output choice.
- ***.ism**. An XML-based *server manifest* file that describes the available bit rates in the encoded presentation. A Smooth Streaming-enabled server uses this file.
- ***.ismc**. An XML-based *client manifest* file that includes important information about the presentation, such as the available bit rates, the codecs that are used, and other information required by Smooth Streaming-compatible clients to view the presentation.

Encoding media files

This section describes how to apply a Smooth Streaming preset—a combination of encoding and playback options that are optimized for Smooth Streaming—to your content after you import it and how to select a Silverlight player template to publish with your encoded media to a Smooth Streaming-enabled server. It does not describe how to prepare your media for encoding. For more information, see Editing your media (http://go.microsoft.com/fwlink/?LinkId=165794).

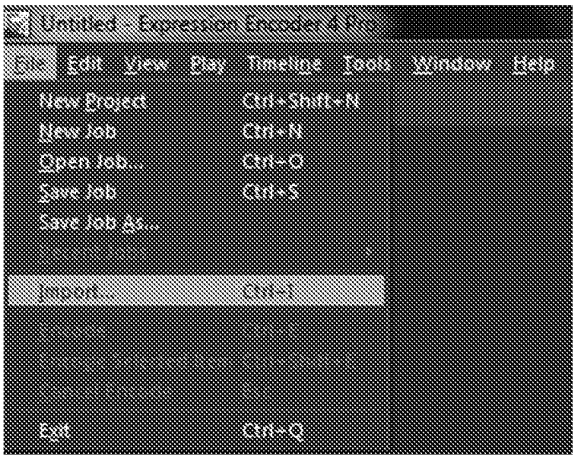
To encode your media, do the following:

1. Start Expression Encoder (click **Start** > **All Programs** > **Microsoft Expression** > **Microsoft Expression Encoder 4**).
2. In the **Load a new project** dialog box, click the **Silverlight Project** workspace, and then click **OK**.

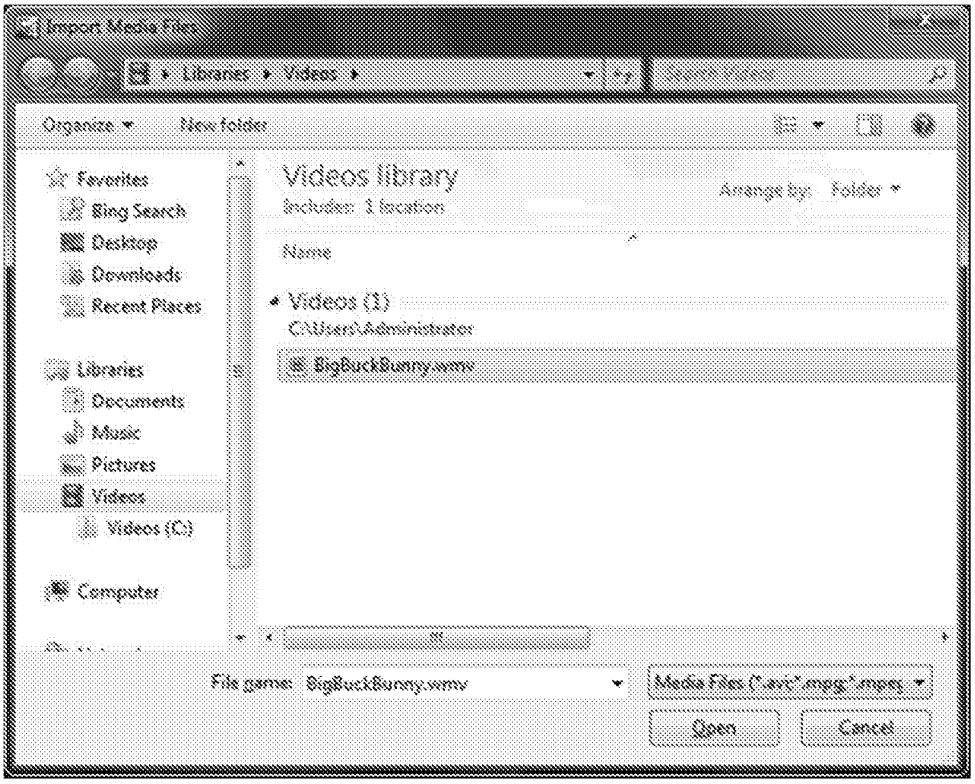


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0898
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3. In Expression Encoder, on the **File** menu, click **Import**.



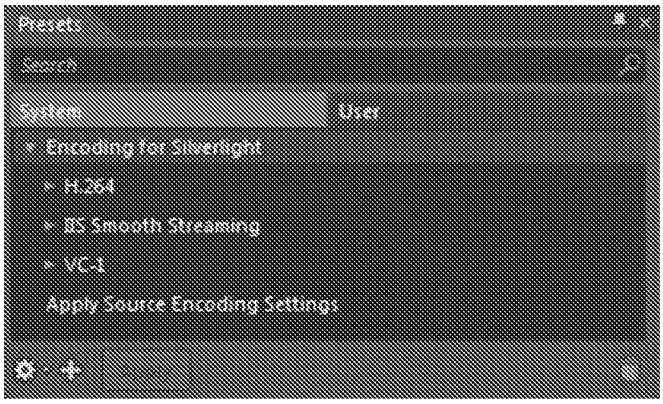
4. In the **Import Media Files** dialog box, select the video that you want to import, and then click **Open**.



5. On the **Presets** tab, apply one of the presets that are optimized for the Smooth Streaming output format. (Note that if you can't see the **Presets** tab, on the **Window** menu in Expression Encoder, click **Presets**.)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0899
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

a. Under **System**, expand the **Encoding for Silverlight** category.

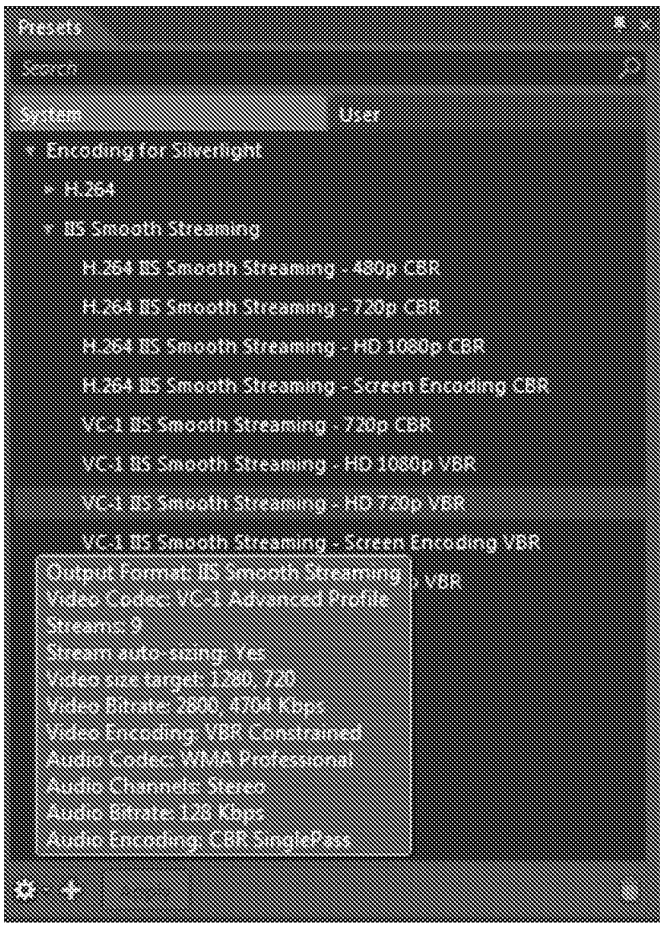


b. Expand the **IIS Smooth Streaming** subcategory to view the list of presets that are optimized for the Smooth Streaming output format.



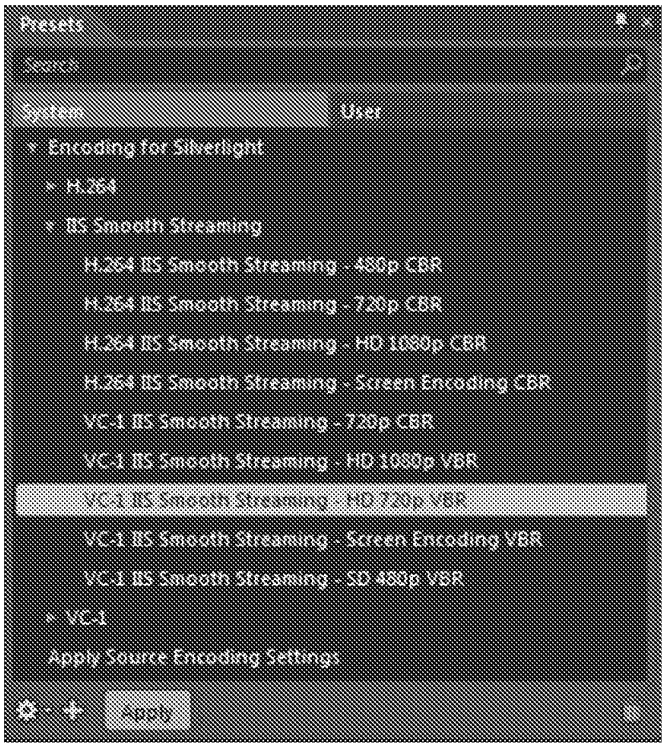
To view parameters for a preset, rest the mouse pointer over your choice. For example, the following parameters are displayed for the preset **VC-1 IIS Smooth Streaming – HD 720p VBR**:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0900
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

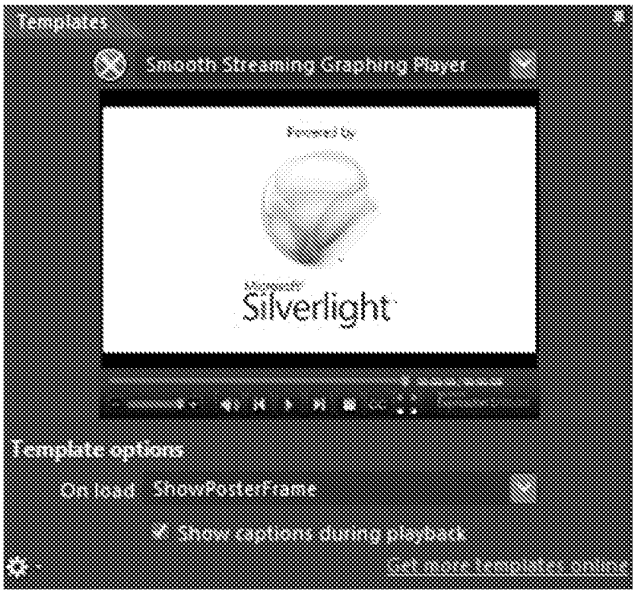


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0901
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

c. Click the preset that you want, and then click **Apply**.



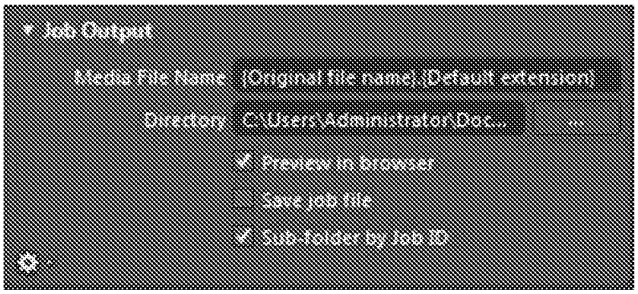
6. On the **Templates** tab, select a Silverlight player template that is included with the encoded output. Templates specify the skin (the visual appearance) that surrounds the video, and can be one of any number of stylized designs that include playback elements.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0902
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

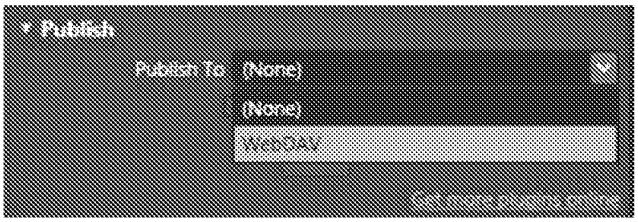
The Silverlight player templates in Expression Encoder may not support some Smooth Streaming playback features. To use a template that fully supports the latest features, in **Template**, select **(None)** so that a Silverlight player template is not included in the encoded output. You can then either use a prepackaged player template that is included with Microsoft Silverlight Media Framework 2.0 (http://go.microsoft.com/fwlink/?LinkID=194570)or create your own player by using IIS Smooth Streaming Client (http://go.microsoft.com/fwlink/?LinkID=194571) APIs. For more information, see Playing Smooth Streams.

7. On the **Output** tab, in the **Job Output** category, select or specify values for job output options.



   - **Media File Name**. Type a file name or choose an expression that extracts the file name from a specified field in the metadata. To choose an expression, right-click the field and choose an expression. By default, the field uses the original file name and the default extension (.ismv) for Smooth Streaming media files.
   - **Directory**. Type the path of the location where you want to save the encoded Smooth Streaming presentation files, or click **Browse for output folder** and browse to a file location.
   - **Sub-folder by Job ID**. Select this option to create a folder in the **Directory** that contains all of the files that are generated. The folder that is created uses the naming convention <*MachineName Date Time*>. Because the encoded output for a Smooth Streaming presentation contains multiple files, you should select this option if you are encoding multiple video files in an encoding job to keep the presentation files separated.

8. (Optional, but recommended) On the **Output** tab, in the **Publish** category, select **WebDAV**.

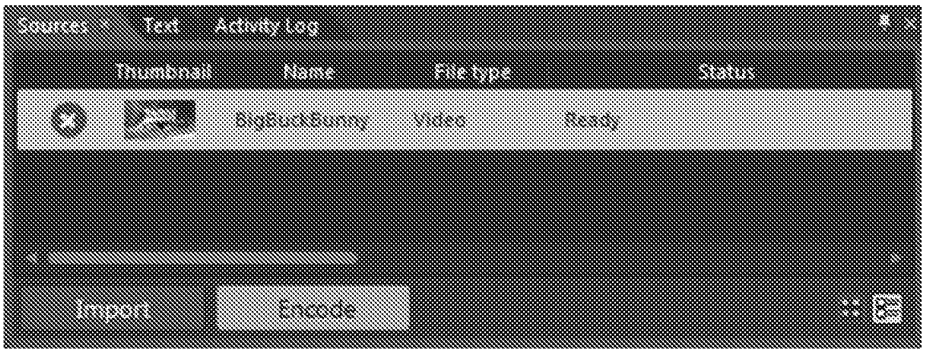


   This setting enables Expression Encoder's WebDAV publishing plug-in to upload the encoded Smooth Streaming presentation files to a WebDAV-enabled Smooth Streaming server using the Web Distributed Authoring and Versioning (WebDAV) protocol.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0903
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Important The **WebDAV** publishing option requires the server to be WebDAV-enabled and also requires you to configure additional settings in Expression Encoder. If you choose to not use WebDAV-publishing to upload the encoded presentation files to the Web server, you can copy the files to the server instead. For more information, see Publishing your media.

9. To encode the video, on the **Sources** tab, click **Encode**.



Publishing your media

After encoding is finished, you can publish the Smooth Streaming presentation files directly from Expression Encoder to a physical directory on a WebDAV-enabled Smooth Streaming Web server. Alternatively, you can create a physical directory on the server in Windows Explorer, and then manually copy the presentation files to the directory.

This section contains the following topics:

- Enabling WebDAV publishing on the Web server
- Publishing the Smooth Streaming presentation to the Web server
- Copying the Smooth Streaming presentation to the Web server

Note

- For more information about how to use WebDAV publishing in Expression Encoder, see Publish a video (http://go.microsoft.com/fwlink/?LinkId=165798).

Enabling WebDAV publishing on the Web server

1. Install and configure the WebDAV Extension for IIS 7 on the Web server. For more information, see Installing and Configuring WebDAV on IIS 7 (http://go.microsoft.com/fwlink/?LinkId=165796).
2. On the Web server, in Server Manager, install one of the following **Security** role services for the **Web Server (IIS)** role if it matches the authentication method that you plan to use for WebDAV publishing:
    - **Basic Authentication**
    - **Windows Authentication**
    - **Digest Authentication**

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0904
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

For more information about installing role services in Server Manager, see Installing Components with Server Manager (http://go.microsoft.com/fwlink/?LinkId=165797).

3. On the Web server, in IIS Manager, click the Web site or directory in the **Connections** pane that you plan to use for delivering your Smooth Streaming presentation.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0905
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

4. On the Home page, double-click the **Authentication** icon, and then enable and configure the **Security** role service that you installed.



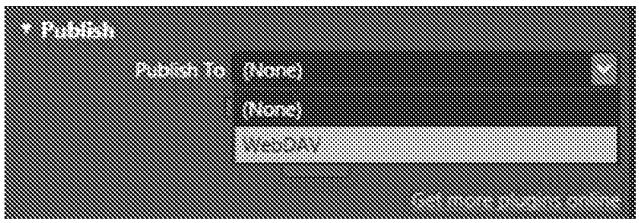
For more information, see Configuring Authentication in IIS 7 (http://go.microsoft.com/fwlink/?LinkId=165799).

Publishing the Smooth Streaming presentation to the Web server

To publish your encoded Smooth Streaming presentation to a Smooth Streaming Web server by using WebDAV, do the following:

1. In Expression Encoder, on the **Output** tab, in the **Publish** category, in **Publish To**, select **WebDAV**.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0906
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



2. In the **Settings** area, specify options for the Web server to which you want to publish the Smooth Streaming presentation:

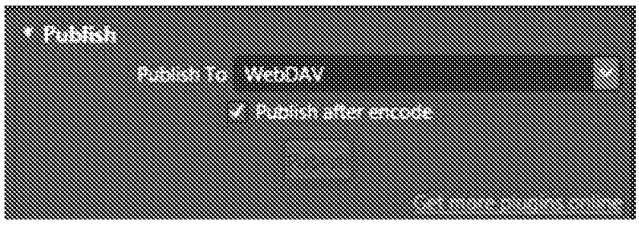


- **Server URL**. Type or select the URL of the server to which you want to publish your media.
- **Use job output folder**. Select this check box if you want to upload the job output folder <*MachineName Date Time*> to the server. Clear this check box so that the Smooth Streaming presentation files within the folder are uploaded to the server, but not the folder itself.
- **Directory Name**. Type the name of a physical directory on the Web server to which you want to publish the folder or files. If the directory is not present on the server, it will be created when you publish your media content. You should use a directory name that is suitable for use in the Web site URL.
- **Authentication**. Select the authentication method that you want to use for publishing the content.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0907
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

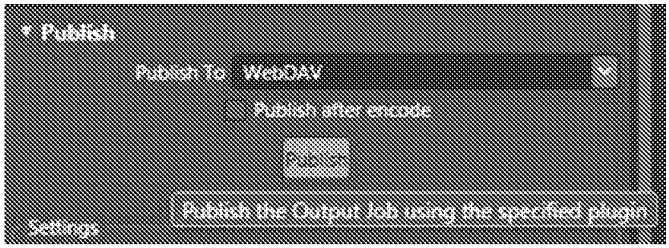
- **Username**/**Password**. Enter the appropriate credentials for the authentication method that you have configured on the Web server.
- **Authenticate**. Click this button to validate the authentication credentials on the server.

3. If you want Expression Encoder to automatically publish your media after encoding is finished, select **Publish after encode**.



-or-

To manually publish the encoded media, clear the **Publish after encode** check box, and instead click the **Publish** button after encoding is finished.



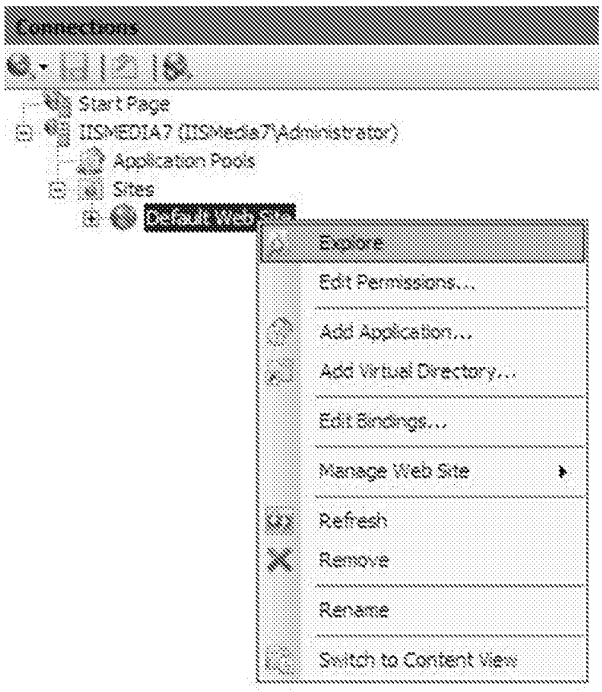
Important Do not click the **Publish** button until you have entered your WebDAV publishing settings, as described in the previous step.

Copying the Smooth Streaming presentation to the Web server

To copy the encoded Smooth Streaming presentation to the Smooth Streaming Web server, do the following:

1. On the encoding workstation, locate the job output folder that contains the encoded Smooth Streaming presentation files. The folder is located in the **Directory** that you specified in the **Job Output** category on the **Output** tab in Expression Encoder. The name of the folder is <*MachineName Date Time*>, which was created when you selected the **Sub-folder by Job ID** check box.
2. On the Web server, in the IIS Manager **Connections** pane, right-click the Web site or directory that you want to use to deliver the Smooth Streaming presentation, and then click **Explore**.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0908
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



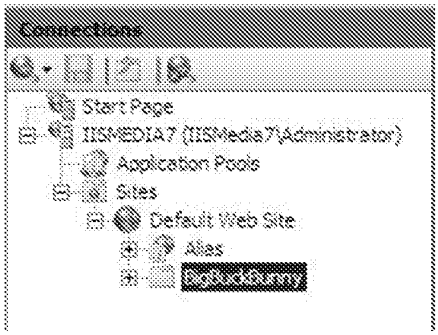
3. Copy the folder <*MachineName Date Time*> from the encoding workstation to the Windows Explorer window that opens. Due to the large size of the folder, you may need to copy it to temporary storage (such as a network share or a high-capacity external hard drive that you can move between computers) first, and then paste it into the window. This adds the <*MachineName Date Time*> physical directory to the Web site.



4. In Windows Explorer, rename the <*MachineName Date Time*> folder to a physical directory name that is more suitable as part of the Web site URL (for example, **BigBuckBunny**).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0909
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5. Close Windows Explorer, and then in IIS Manager, refresh the Web site or directory to view the new physical directory name.



Encoding Live Smooth Streams Using Expression Encoder

Live Smooth Streaming encoders generate multiple streams, each containing MPEG-4 (MP4) fragments encoded at different bit rates. As in the encoded output for on-demand Smooth Streaming presentations, there can be multiple video and audio streams, depending on the encoder configuration. However, instead of storing the fragments sequentially in MP4-container files, the encoder delivers them directly to Live Smooth Streaming publishing points on Web servers. The metadata encoded in the Live Smooth Streaming output also differs from that encoded in on-demand Smooth Streaming files in that additional information is included in the encoded output to support Live Smooth Streaming features such as DVR-style client interaction, archiving by publishing points, and so on. Live Smooth Streaming encoders also do not output client manifest files as on-demand Smooth Streaming encoders do. Instead, the Live Smooth Streaming publishing point on the server generates a manifest for clients, based on information provided to it by the encoder.

When a Live Smooth Streaming publishing point receives a request for media, it retrieves virtual fragments from the most appropriate stream in the live broadcast for the conditions at any given time in order to deliver the best content possible to the client.

Encoding a live broadcast

This section describes how to create a Live Smooth Streaming broadcast in Expression Encoder 4 Pro and push it to a Live Smooth Streaming publishing point. The procedure in this section outlines a general workflow, with details only on the specific tasks that you need to create Live Smooth Streaming output. For detailed information about how to create a live broadcast from various live and file-based sources in Expression Encoder 4, including how to assemble a complete program, encode it, encrypt it with Digital Rights Management (DRM), and instantly distribute it to your audience using Smooth Streaming technology, see Creating a Live Broadcasting Project (http://go.microsoft.com/fwlink/?LinkId=199825).

Before you proceed, make sure that the following prerequisite conditions are fulfilled:

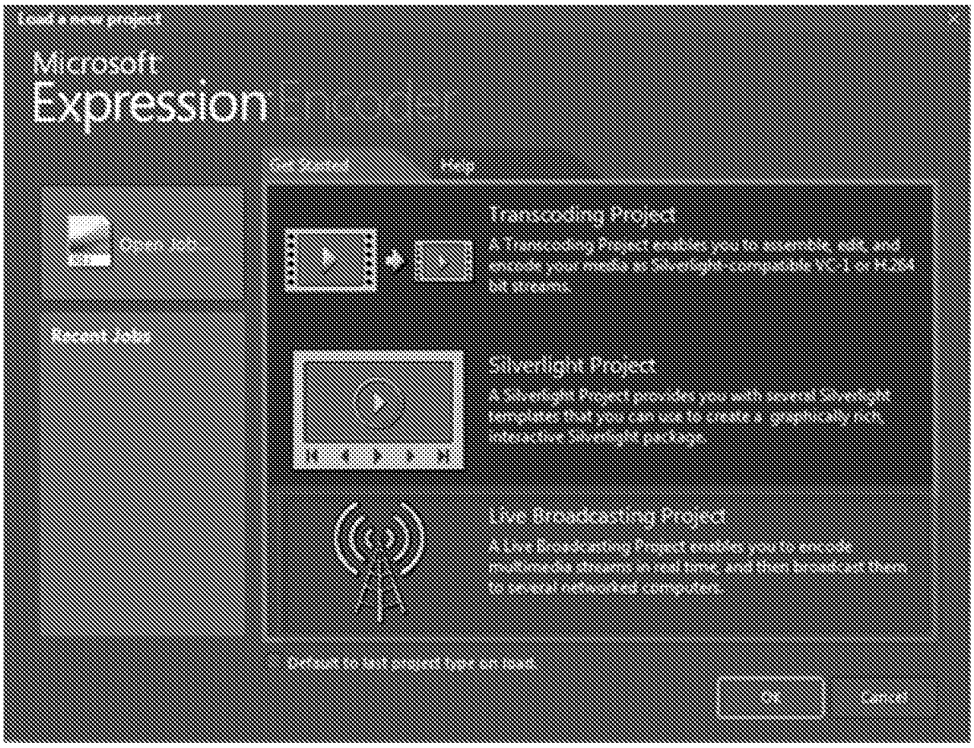
- A Live Smooth Streaming publishing point is configured and started on the Web server. For more information, see Serving Live Smooth Streams.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0910
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- A Web page is updated so that clients can access the Live Smooth Streaming presentation. For more information, see Enabling playback of Smooth Streams.

To encode a live broadcast, do the following:

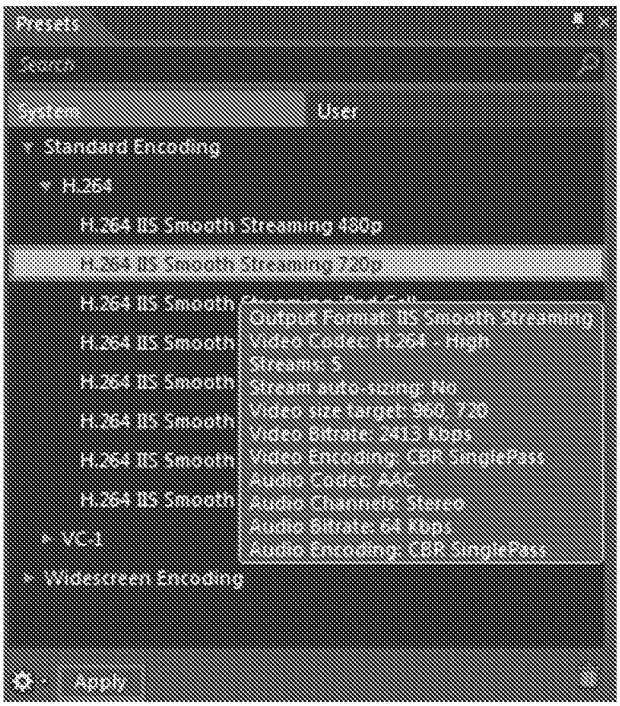
1. Start Expression Encoder (click **Start** > **All Programs** > **Microsoft Expression** > **Microsoft Expression Encoder 4**).
2. In the **Load a new project** dialog box, select **Live Broadcasting Project**, and then click **OK**.



3. Set up the source to use for your live broadcast. For file sources, you can use audio or video sources in any file format that Expression Encoder can import. For live sources, you can connect multiple camera devices, such as USB webcams or FireWire (IEEE 1394) digital video cameras. Although you can connect multiple live sources, you can stream only one at a time. For more information about setting up sources for live broadcasts, see the following topics:
   - Set File Sources (http://go.microsoft.com/fwlink/?LinkId=199826)
   - Set Live Sources (http://go.microsoft.com/fwlink/?LinkId=199827)

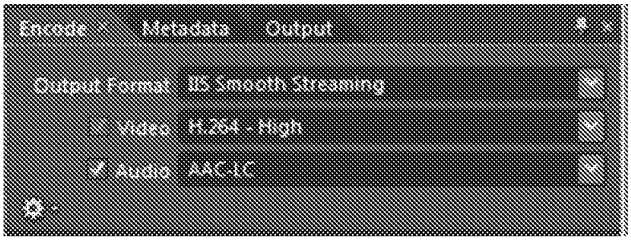
4. On the **Presets** tab, choose an encoding preset that supports the bit rates and encoding requirements for your Live Smooth Streaming scenario. Be sure to choose a preset with **IIS Smooth Streaming** in the preset name, and then click **Apply**.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0911
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



To find out more information about a preset, such as the numbers of streams in the output and the codecs used, rest your mouse pointer over a preset name. The preset information is displayed in a box.

Note that clicking the **Apply** button automatically updates the **Output Format**, **Video**, and **Audio** settings on the **Encode** tab with the values used by the selected preset.

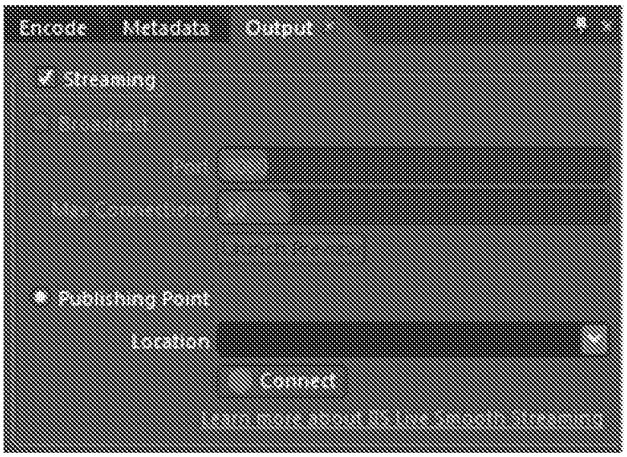


You can choose to not select a preset and specify your own values for these settings on the **Encode** tab instead. For more information about how to do this, see the following topics:

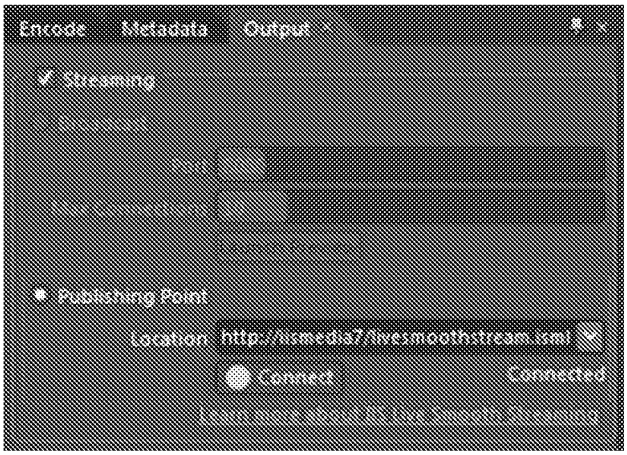
- Set Output Format (http://go.microsoft.com/fwlink/?LinkId=199828)
- Set Video options (http://go.microsoft.com/fwlink/?LinkId=199829)
- Set Audio options (http://go.microsoft.com/fwlink/?LinkId=199830)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0912
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5. On the **Output** tab, select the **Streaming** check box to activate streaming.



With Streaming selected on the **Output** tab, your only choice for publishing your content is to use the **Publishing Point** option, which is automatically selected (If it isn't selected, select it). In the **Location** box, type the URL of the Live Smooth Streaming publishing point that you want to publish your live broadcast to.



URL syntax:
http://*ServerName*/*SiteName*/*DirectoryName*/*PublishingPointDefinitionFileName*.isml

Note that after you specify publishing point URL in the **Location** box, you can click the **Connect** button to test the connection to the publishing point. The publishing point must be started for the connection to be validated. For more information about how to configure and start a Live Smooth Streaming publishing point, see Serving Live Smooth Streams.

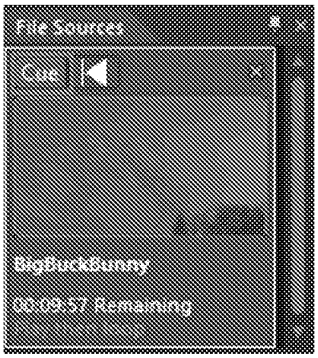
6. If you want to add Digital Rights Management (DRM) to your content, select the **Digital Rights Management** check box on the **Output** tab, and then type the **License URL** (the server that

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0913
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

supplies the license), the **Key ID**, and the **Key Seed** (available from the license provider). For more information, see Set DRM options (http://go.microsoft.com/fwlink/?LinkId=199832).

Note that the license provided for a Live Smooth Streaming session is non-persistent, meaning that after the user closes the client application being used to view the Web broadcast, the license expires.

7. Click **Cue** to enable your source for broadcasting (a cued file source is shown in the following figure)...



...and then click **Start** to begin your live broadcast.



As the broadcast runs, you can monitor statistics and connections data in the corresponding panels. For more information about how to monitor this data, see the following topics:

- Using the Statistics panel (http://go.microsoft.com/fwlink/?LinkId=199834)
- Using the Connections panel (http://go.microsoft.com/fwlink/?LinkId=199835)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0914
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Serving Smooth Streams

The Smooth Streaming extensions in IIS Media Services enable adaptive streaming of media from a Web server to Smooth Streaming-compatible clients. Smooth Streaming provides a high-quality viewing experience that scales massively on content distribution networks, making full-HD (1080p) media experiences a reality.

Smooth Streaming uses HTTP to take advantage of the scale of existing HTTP networks, keeping content closer to the audience. Unlike traditional streaming, Smooth Streaming uses standard HTTP to deliver cacheable MPEG-4 (MP4) fragments, eliminating the need for provisioning a streaming server network with the required number of stateful streaming sessions per server to scale the delivery of media to the expected audience. This means that your existing HTTP infrastructure can be used to dramatically increase media availability, and you do not have worry about how much server capacity you can dedicate to an event, as you would on a more resource-constrained traditional streaming network.

For the audience, the improved video viewing experience brings the reliability and quality of television to their favorite video Web sites. Today, content owners largely target video quality for the lowest common denominator of their audience, even though many of their viewers have high-quality, high-speed connections. Smooth Streaming dynamically adapts stream quality based on a client's momentary connection speed so that users with high-bandwidth connections can experience near-HD or full-HD quality streaming, while users with lower bandwidth connections receive the highest possible bit rate based on their current connection speed and video processing capability.

Note

- You can use caching servers to store the downloadable Smooth Streaming MP4 fragments requested by clients to store media content closer to end users. This can further increase the scale of your existing HTTP infrastructure and increase the availability of media events to clients. For popular events, caching servers can store the fragments requested by clients, and then serve those fragments directly to each subsequent user until the content expires. For more information, see Edge Caching for Media Delivery (http://go.microsoft.com/fwlink/?LinkId=166161).

System Requirements

The following prerequisites must be fulfilled to use the Smooth Streaming extensions:

- You must use IIS 7 on one of the following operating systems:
  - Windows Server 2008 R2
  - Windows Server® 2008
  - Windows 7
  - Windows Vista with Service Pack 1 (SP1)
- To manage the Smooth Streaming extensions by using the IIS Manager user interface (UI), the **IIS Management Console** for IIS must be installed. In Windows Server 2008 R2 or Windows Server 2008, you can install the **IIS Management Console** role service for the **Web Server (IIS)**

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0915
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

role in Server Manager. For more information, see Installing Components with Server Manager (http://go.microsoft.com/fwlink/?LinkId=165797).

In Windows 7 or Windows Vista, turn on the **IIS Management Console** feature for IIS. For more information, see Turn Windows features on or off (http://go.microsoft.com/fwlink/?LinkId=166162).

- A Web server that you configure as an on-demand Smooth Streaming server by installing the IIS Smooth Streaming extension can accept an uploaded Smooth Streaming presentation that is published directly from Expression Encoder using WebDAV. To enable the upload, the WebDAV Extension for IIS 7 must be installed on the server. In addition, because WebDAV publishing requires authentication, you must install the **Security** role service for the **Web Server (IIS)** role that matches the authentication method that you use for WebDAV publishing. For more information, see Enabling WebDAV publishing on the Web server.
- A Web server that you configure as a Live Smooth Streaming server by installing the IIS Live Smooth Streaming extension may not work correctly if the **ASP.NET** role service for the **Web Server (IIS)** role is installed. For more information about removing this role service, see Installing Components with Server Manager (http://go.microsoft.com/fwlink/?LinkId=165797).

Installing the Smooth Streaming Extensions

The Smooth Streaming extensions for IIS 7 are included in the following versions of IIS Media Services:

- **IIS Media Services 3.0**. This version of IIS Media Services contains a "Go Live" license, which means that you can use the software in live operating environments. It is this version of the software that is documented in this article.
- **IIS Media Services 4.0 Beta 1**. This version of IIS Media Services is not licensed for use in live operating environments. For more information about the new features in the Beta release, which you can use in test environments, see the IIS Media Services Web page (http://go.microsoft.com/fwlink/?LinkID=166171).

To install the IIS Smooth Streaming extensions, do the following:

1. To ensure successful installation of the Smooth Streaming extensions, close IIS Manager if it is running.
2. Go to the Microsoft Web Platform Installer Download Page (http://go.microsoft.com/fwlink/?LinkId=166170) and click **Download It Now** to download and run Microsoft Web Platform Installer.
3. In the lower-left corner of the **Web Platform Installer** dialog box, click **Options**.
4. In the **Change Options** dialog box, under **Display additional scenarios**, make sure that the **Media** check box is selected, and then click **OK**.
5. Back in the **Web Platform Installer** dialog box, on the **Media Platform** tab, in the **Host, Deliver, and Scale Using IIS Media Services** area, click **Customize**.
6. Select **IIS Media Services 3.0**, and then click **Install**.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0916
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Serving On-Demand Smooth Streams

The Smooth Streaming extension for IIS 7 works with contiguous files stored in a fragmented MPEG-4 (MP4) media container format, which are created by encoders that use a Smooth Streaming profile to encode the video source. A Smooth Streaming video presentation created by an encoder typically contains the following:

- Multiple video files (with .ismv file name extensions) encoded with different quality levels (bit rates) and resolutions. If the video source contains audio, one of the .ismv files contains the audio stream, typically encoded at 64 kilobits per second (Kbps).
- XML manifest files that enable clients to use Smooth Streaming heuristics, a stream-switching logic that obtains the best quality video optimized for the client environment at all times.

Encoders may also include a Silverlight player template with the presentation files in the encoded output. If a template is not included, you can either use a prepackaged player template that is included with Microsoft Silverlight Media Framework 2.0 (http://go.microsoft.com/fwlink/?LinkID=194570)or create your own player by using IIS Smooth Streaming Client (http://go.microsoft.com/fwlink/?LinkID=194571) APIs. For more information, see Playing Smooth Streams.

For more information about encoders that support on-demand Smooth Streaming output, see the list of our Encoding Partners (http://go.microsoft.com/fwlink/?LinkId=165785).

When a client makes a request for a Smooth Streaming presentation, the Web server delivers an XML manifest that describes the available quality levels. The client then uses the Smooth Streaming heuristics to determine the best file from which to request a media fragment at that instant. The Smooth Streaming heuristics continuously monitor the environment and at frequent intervals (typically 2 seconds), request the next contiguous MP4 fragment in the file (if available bandwidth and video playback resources have not changed) or a contiguous MP4 fragment from a different file (if available bandwidth and video playback resources have changed). IIS Smooth Streaming dynamically creates the MP4 fragment from the appropriate video source file and delivers it seamlessly to the client. This process continues until the client stops playback or the end of the file is reached, ensuring that the client has the best possible viewing experience without any buffering or stuttering. Clients that host Smooth Streaming heuristics include Microsoft Silverlight (http://go.microsoft.com/fwlink/?LinkId=165789).

This section contains the following topics:

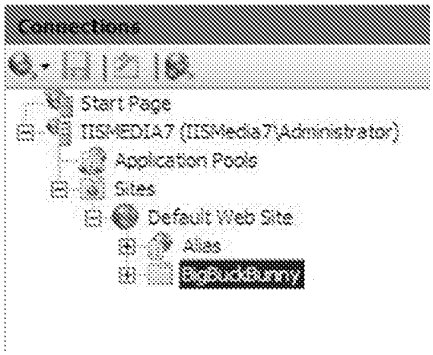
- Serving an on-demand Smooth Streaming presentation
- Viewing Smooth Streaming presentation details
- Editing a Smooth Streaming presentation
- Copying a Smooth Streaming presentation
- Configuring advanced settings

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0917
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Serving an on-demand Smooth Streaming presentation

To serve an on-demand Smooth Streaming presentation to clients, do the following:

1. If the Smooth Streaming presentation does not contain a Silverlight player template, download the Smooth Streaming Silverlight player template for use by clients. For more information, see Playing Smooth Streams. If you are not sure if the presentation contains a template, see Viewing Smooth Streaming presentation details.
2. If the Smooth Streaming presentation contains a Silverlight player template, no additional configuration steps are required to deliver the presentation. Because IIS Smooth Streaming is enabled by default in IIS Manager, clients can connect to a Smooth Streaming presentation by specifying the presentation URL, including the directory path if required. In the example in the previous section, the Smooth Streaming presentation files were published to the **BigBuckBunny** physical directory on the Web server.



   In this example, clients can stream the presentation by using the following URL: **http://iismedia7/BigBuckBunny/default.html**.

   -or-

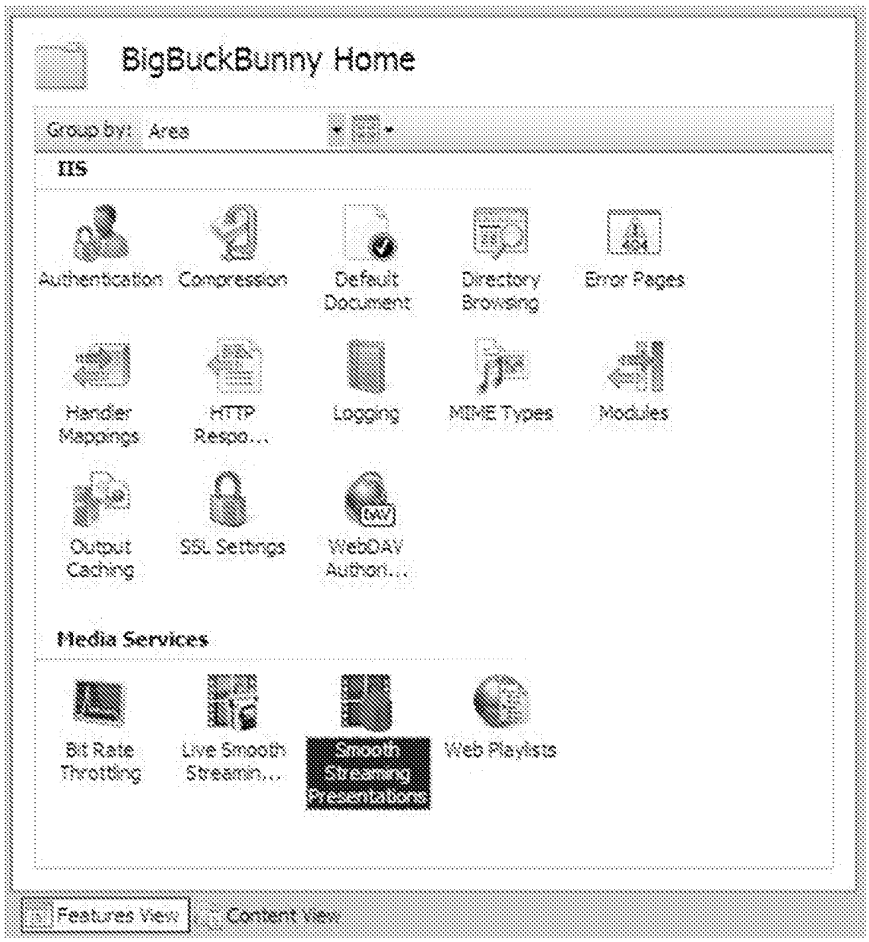
   If you specify that **default.html** be returned when clients do not specify a specific file in the directory, clients can stream the presentation by using the following URL: **http://iismedia7/BigBuckBunny/**. For more information, see Configuring Default Documents in IIS 7 (http://go.microsoft.com/fwlink/?LinkId=166163).

Viewing Smooth Streaming presentation details

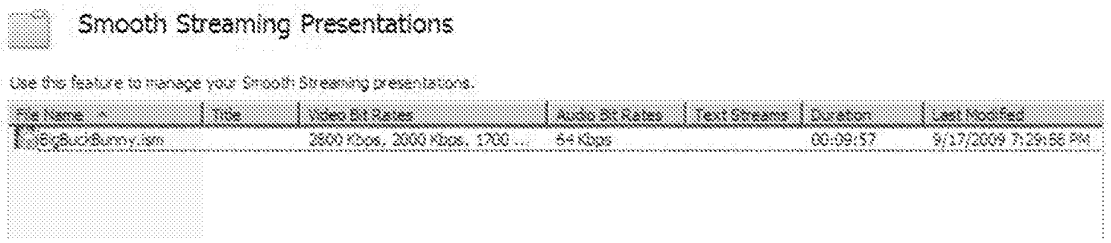
To view Smooth Streaming presentation details, do the following:

1. In the **Connections** pane, click the Smooth Streaming presentation directory, and then in the directory **Home** page, double-click the **Smooth Streaming Presentations** icon.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0918
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



The following presentation details are displayed in **Features View**:

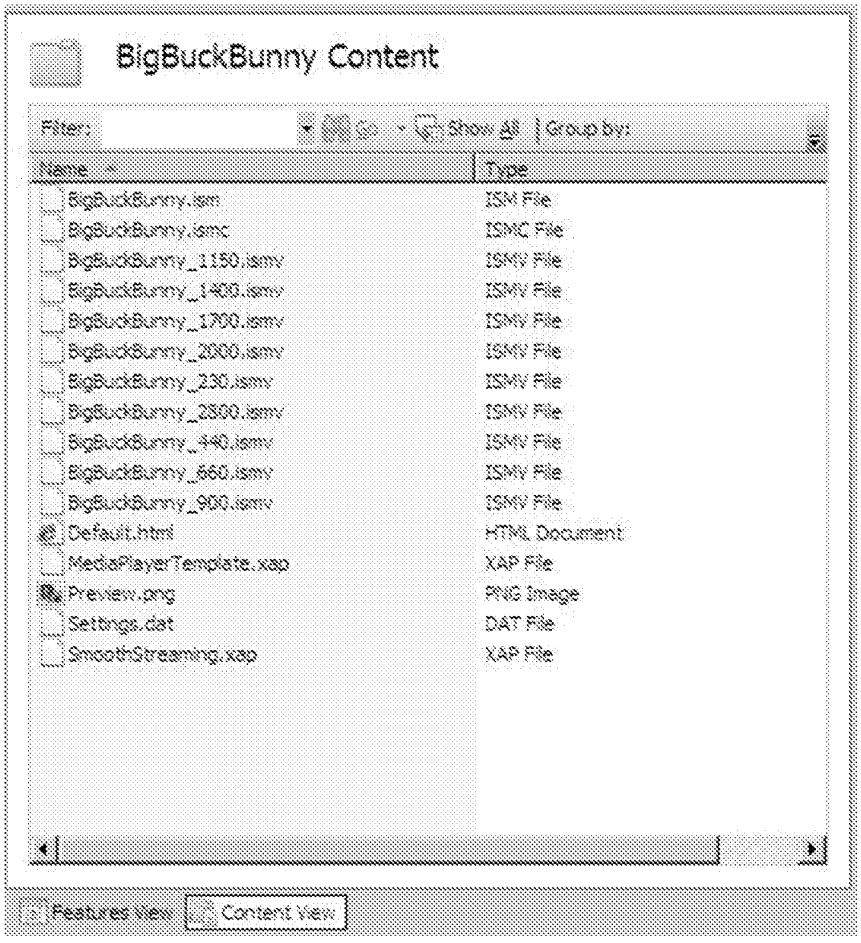


- **File Name**. Displays the name of the presentation server manifest (a file that has an .ism file name extension).
- **Title**. Displays the descriptive title of the presentation, if one is specified.
- **Video Bit Rates**. Displays the available video bit rates, in kilobits per second (Kbps), which are included in the presentation.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0919
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- **Audio Bit Bates**. Displays the available audio bit rates, in kilobits per second (Kbps), which are included in the presentation.
- **Text Streams**. Displays the number of text streams that are included in the presentation.
- **Duration**. Displays the length of the presentation.
- **Last Modified**. Displays the date and time when the presentation was last modified.

2. To view the Smooth Streaming presentation files that are created by the encoder, click **Content View** below the **Home** page.



The files shown in the figure include Smooth Streaming presentation files (.ism, .ismc, and *.ismv) and Silverlight player template files (Default.html, MediaPlayerTemplate.xap, and SmoothStreaming.xap). If your directory does not contain the template files, you can download a sample Smooth Streaming player to the directory. For more information, see Playing Smooth Streams.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0920
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Editing a Smooth Streaming presentation

Editing a Smooth Streaming presentation means that you change the bit rates that are available in the presentation by removing references to them from the presentation manifest (.ism and .ismc) files. Removing bit rates makes them unavailable for streaming but does not remove the corresponding .ismv files. Some of the reasons why you might want to remove available bit rates include:

- Your Web server struggles to deliver the higher bit rate streams in a Smooth Streaming presentation due to server bandwidth or resource limitations. You want to remove one or more of the higher bit rates from the presentation.
- Clients connect to your server through a wide-area-network (WAN) and cannot view the higher bit rate streams. You want to remove one or more of the higher bit rates from the presentation.
- Clients can view your content through multiple channels (for example, you make HD video available through a premium channel and offer SD versions of the video as free samples). You want to remove one or more of the higher bit rate streams that are available to create an SD version of an HD presentation.

Undoubtedly, there are many other reasons why you might want to change the available bit rates in a presentation.

Important

- You should create a copy of the presentation before you remove bit rates from it so that you have a copy of the original presentation for later use. For example, if you want to create an SD version of your HD presentation, you can remove bit rates from one of the copies in order to have two versions of the presentation: an SD version in which some of the higher bit rates are removed and an HD version in which all of the originally encoded bit rates are available. For more information, see Copying a Smooth Streaming presentation.

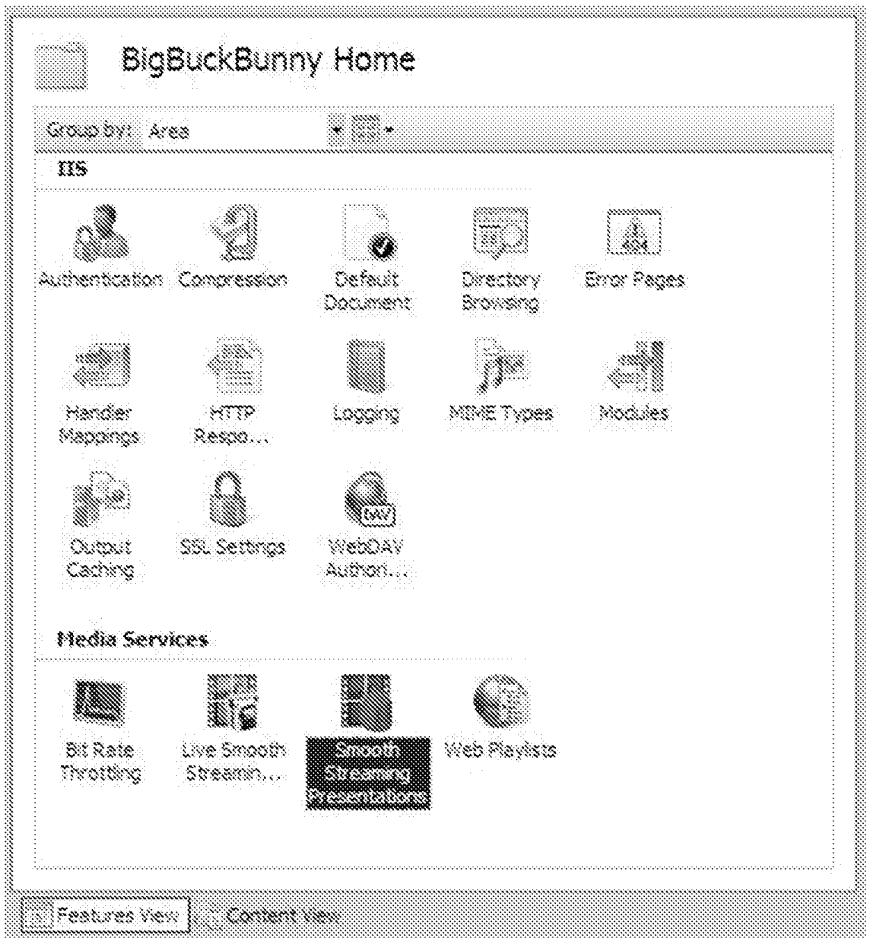
Note

- For more information about how to configure settings that increase Web server performance, see Configuring advanced settings.

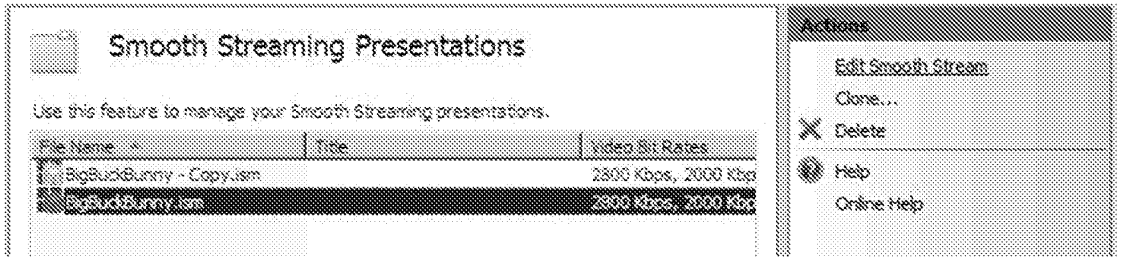
To remove a bit rate from a Smooth Streaming presentation, do the following:

1. In the **Connections** pane, click the Smooth Streaming presentation directory, and then in the directory **Home** page, double-click the **Smooth Streaming Presentations** icon.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0921
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

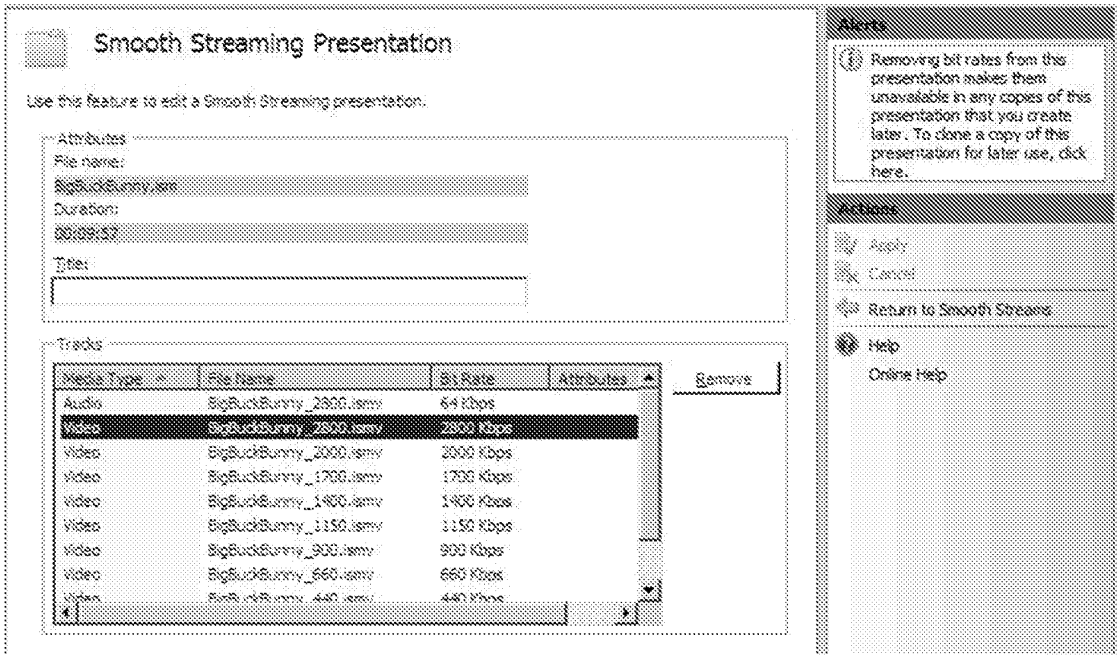


2. In the **Smooth Streaming Presentations** page, select the presentation, and then, in the **Actions** pane, click **Edit Smooth Stream**.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0922
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3. In the **Smooth Streaming Presentation** page , do the following to remove (for example) the highest bit rate from the presentation:



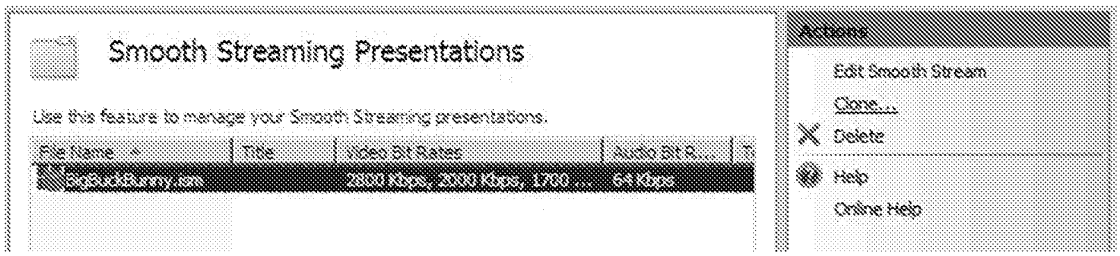
   a. Select the video track with the highest bit rate value.
   b. Click **Remove**.
   c. In the **Actions** pane, click **Apply**.

Copying a Smooth Streaming presentation

You can clone a presentation to create copies of the presentation manifest (.ism and .ismc) files. Cloning does not create copies of the media (.ismv) files. The original and cloned presentations share the same set of media files. Cloning is useful if you want to make different bit rates available to clients from the same presentation. For example, if you want to make only the higher bit rates available through a premium channel, you can make the higher bit rates unavailable in a copy of the presentation that you offer as a free sample. For more information, see Editing a Smooth Streaming presentation.

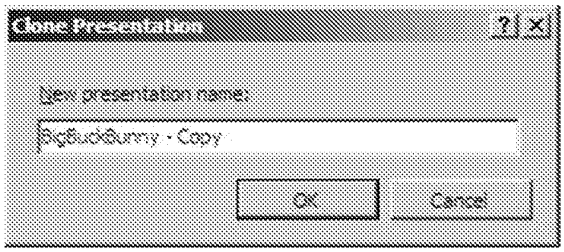
To create a copy of a presentation, do the following:

1. Select your presentation in the **Smooth Streaming Presentations** page, and then in the **Actions** pane, click **Clone**.

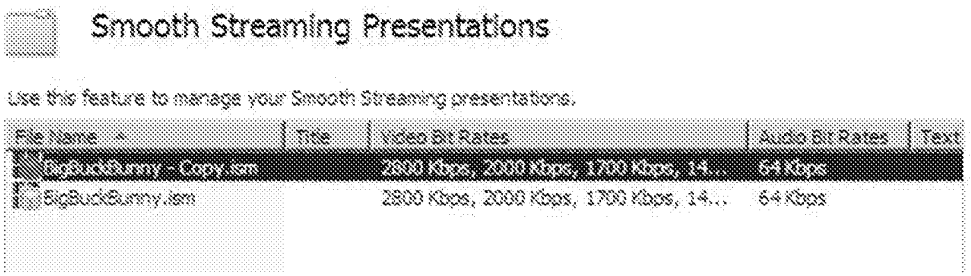


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0923
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2. In the **Clone Presentation** dialog box, enter a new name for your cloned presentation, and then click **OK**.

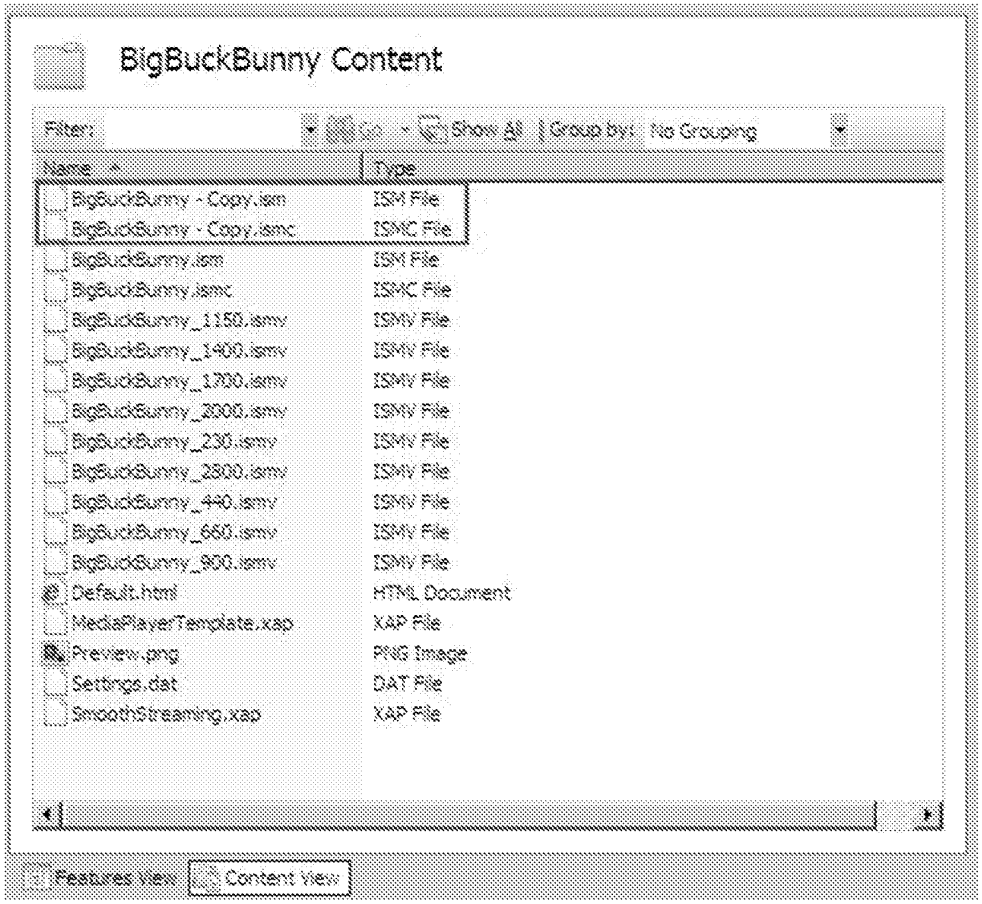


3. The cloned presentation is displayed in the **Smooth Streaming Presentations** page.



4. In **Content View**, new .ism and .ismc files have been created for the cloned presentation, with the original and cloned presentations sharing the same media files.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0924
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



Configuring advanced settings

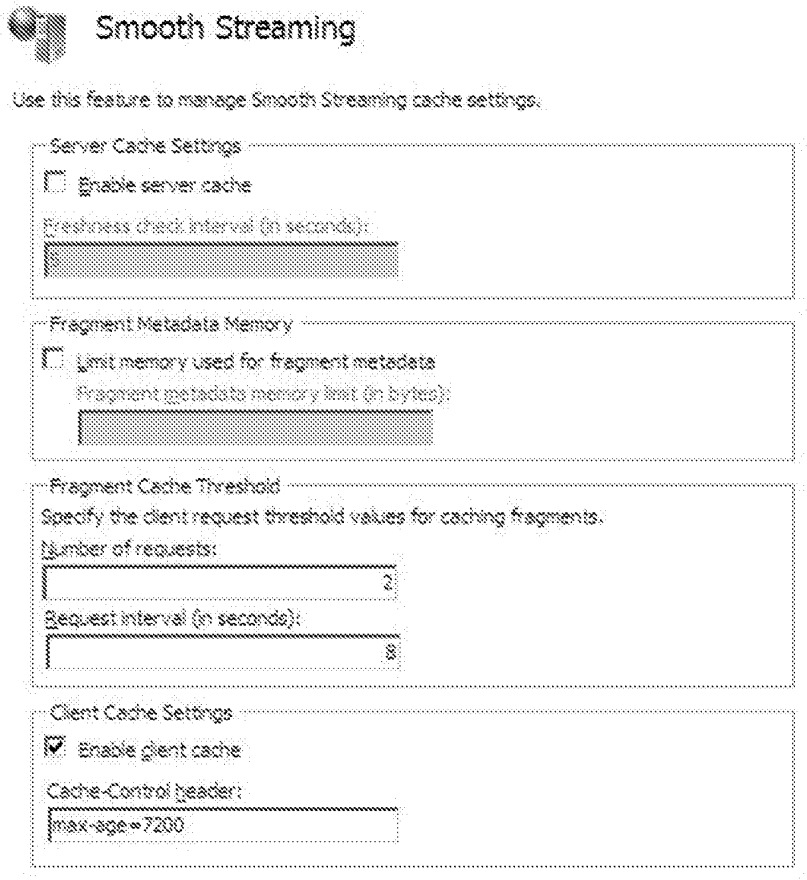
The **Smooth Streaming** feature page contains settings that you can configure to increase Web server performance, reduce latency in the HTTP response to clients, and control how clients can cache the Smooth Streaming content that they receive.

To configure these settings, do the following:

1. In the **Connections** pane, click the server name, and then in the server **Home** page, double-click the **Smooth Streaming Presentations** icon.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0925
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2. Configure the following settings in the **Smooth Streaming** feature page, and then in the **Actions** pane, click **Apply**.



Server Cache Settings

Use the **Server Cache Settings** area to configure output caching to improve Web server performance and reduce latency in the HTTP response to clients.

- **Enable server cache**. Enables caching of Smooth Streaming presentation data (metadata and content indexes) using the **Output Caching** feature settings in IIS Manager. For more information about how to configure output caching, see Configuring Output Caching in IIS 7 (http://go.microsoft.com/fwlink/?LinkId=166164).

  Note that this option does not enable caching of Smooth Streaming video fragments. To cache fragments, you must specify threshold values in the **Fragment Cache Threshold** area of the feature page.

- **Freshness check interval**. Enter a time, in seconds, that specifies how often cached Smooth Streaming data should be compared to the original data sources in the file

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0926
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

system. If the comparison reveals that the original data sources have been updated, the cached data is deleted. By default, the comparison is performed every 5 seconds.

Fragment Metadata Memory

Use the **Fragment Metadata Memory** area to specify the amount of server memory to use for caching Smooth Streaming fragment data (metadata and indexes). When clients request a Smooth Streaming presentation, fragment data can be stored in memory. This enables the Smooth Streaming feature to quickly retrieve MP4 video fragments from the file system for subsequent client requests. The settings enforce a least-recently-used (LRU) algorithm so that the oldest Smooth Streaming fragment metadata is overwritten when the cache is full.

Note that these settings do not enable caching of Smooth Streaming video fragments. To cache fragments, you must specify threshold values in the **Fragment Cache Threshold** area on this page.

- **Limit memory used for fragment metadata**. Clear this check box to use the server's default cache size for caching Smooth Streaming fragment metadata. If you want to specify the amount of memory to use, select this check box, and then enter a value in **Fragment metadata memory limit**, in bytes.

Fragment Cache Threshold

Use the **Fragment Cache Threshold** area to create a fragment-caching threshold rule that specifies which video fragments are cached. Cached fragments do not have to be reprocessed from the Smooth Streaming source files before they are delivered to clients. This can improve Web server performance and reduce response latency.

You should specify threshold values that exclude less frequently requested fragments from being cached. If you create a threshold value that is too low (for example, by requiring that only a few requests for fragments occur during a lengthy request interval), the increased disk I/O operations that occur when Smooth Streaming tries to cache lots of fragments can decrease performance.

- **Number of requests**. Enter the number of times a fragment must be requested within the Request interval before it is cached. Note that if you set the value to **1**, all requested fragments are cached, which can significantly impact performance.
- **Request interval**. Enter the time, in seconds, during which requests for a fragment are accumulated. If the total accumulation of requests for a particular fragment exceeds the **Number of requests** value within this request interval, then the fragment is cached.

Client Cache Settings

Use the **Client Cache Settings** area to control how client browsers and proxy servers cache the Smooth Streaming content returned by the Web server.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0927
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- **Enable client cache**. Enables caching of content by clients, according to caching rules specified in **Cache-Control header**.
- **Cache-Control header**. Enter a Cache-Control header directive to be included in the HTTP header response sent to clients. The Cache-Control header directive specifies the conditions and restrictions for caching Smooth Streaming content on the client. For a list of valid directives that you can enter, see Cache-Control (http://go.microsoft.com/fwlink/?LinkId=166165).

Serving Live Smooth Streams

The Live Smooth Streaming extension for IIS 7 works with the fragmented MPEG-4 (MP4) output from encoders that use a Live Smooth Streaming profile to encode the video source. A Live Smooth Streaming video presentation created by an encoder contains the following:

- Video and audio MP4 fragments encoded as separate streams with different quality levels (bit rates) and resolutions.
- Metadata that enable Live Smooth Streaming publishing points to create XML manifest files. These files allow clients to use Smooth Streaming heuristics, a stream-switching logic that obtains the best quality stream optimized for the client environment at all times.

For more information about encoders that support Live Smooth Streaming output, see the list of our Encoding Partners (http://go.microsoft.com/fwlink/?LinkId=165785).

Live Smooth Streaming publishing points on the Web Server with IIS Live Smooth Streaming installed are used to deliver the fragmented MP4 output from encoders to Smooth Streaming-enabled clients. Publishing points can also distribute the live streams from an encoder to Live Smooth Streaming publishing points on other servers in order to spread the load across a streaming server network.

When a client makes a request for a Live Smooth Streaming presentation, the publishing point on the Web server delivers an XML manifest that describes the available quality levels. The client then uses the Smooth Streaming heuristics to determine the best stream from which to request a media fragment at that instant. The Smooth Streaming heuristics continuously monitor the environment and at frequent intervals (typically 2 seconds), request the next contiguous MP4 fragment in the live stream (if available bandwidth and video-processing power have not changed) or a contiguous MP4 fragment from a different stream (if available bandwidth and video-processing power have changed). This process continues until the client stops playback or the end of the stream is reached, ensuring that the client has the best possible viewing experience without any buffering or stuttering. Clients that host Smooth Streaming heuristics include Microsoft Silverlight (http://go.microsoft.com/fwlink/?LinkID=165789).

This section contains the following topics:

- Creating a publishing point
- About publishing point roles
- Monitoring publishing points

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0928
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Managing credentials
- Configuring advanced settings

Creating a publishing point

Live Smooth Streaming publishing points control the flow of live streams from an encoder to Smooth Streaming-compatible clients or to Live Smooth Streaming publishing on other servers, which in turn can service client requests for live streams to distribute the load across a streaming server network. On the Web server, the publishing point settings are stored in a file with an .isml file name extension.

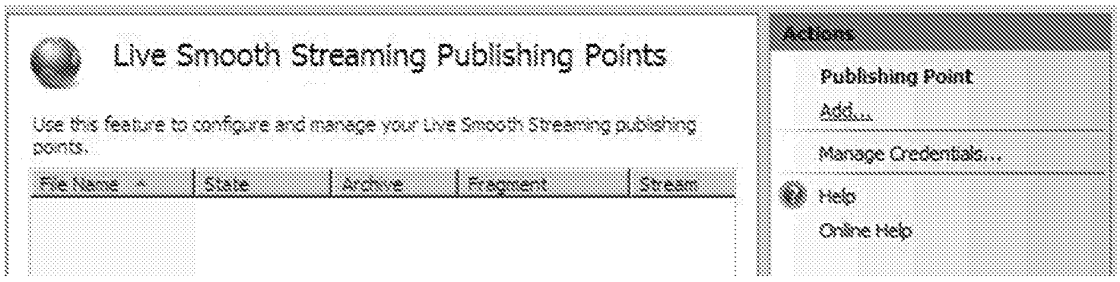
To create a publishing point, do the following:

1. In the **Connections** pane, click the Web site or directory that you want to use for delivering live streams, and then in the **Home** page, double-click the **Live Smooth Streaming Publishing Points** icon.

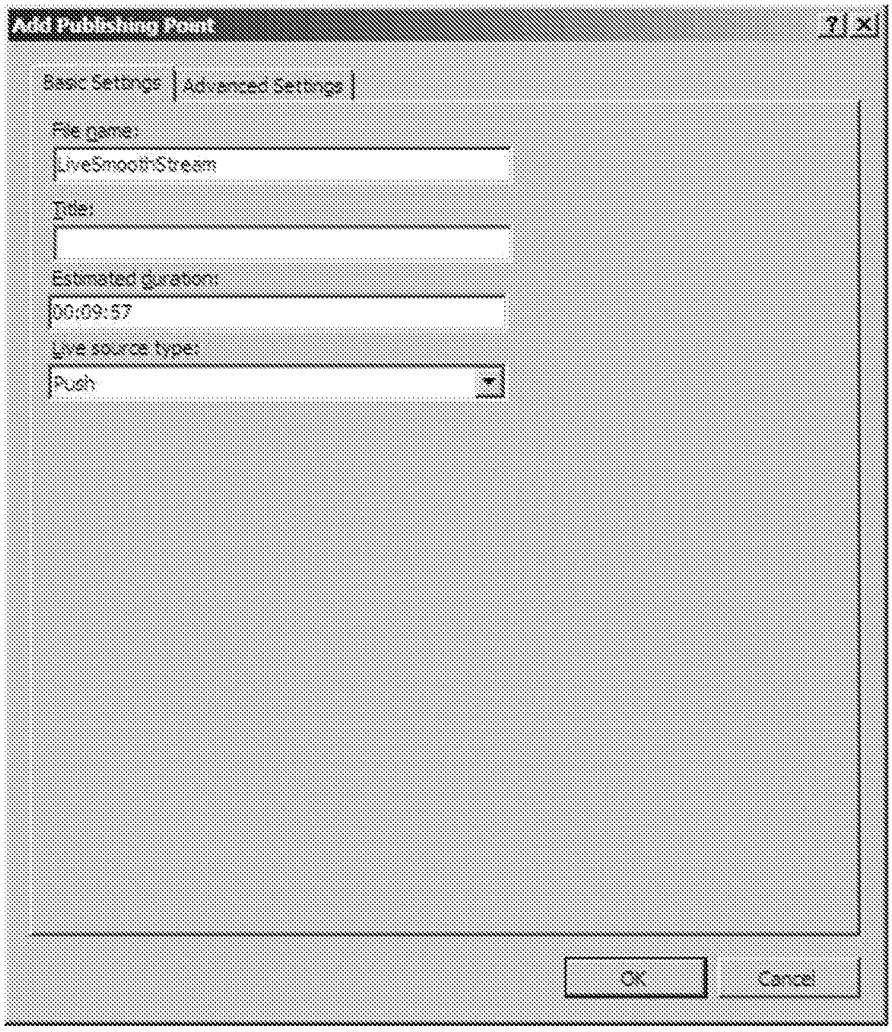


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0929
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2. In the **Live Smooth Streaming Publishing Points** feature page, in the **Actions** pane, click **Add**.



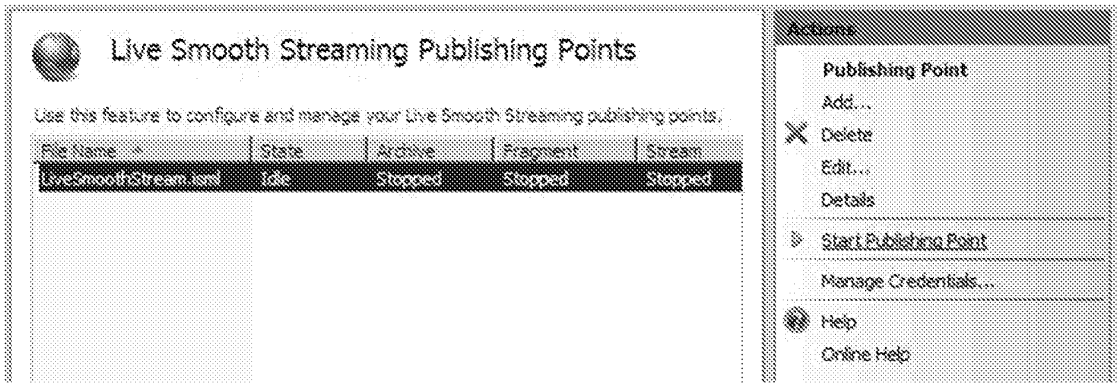
3. In the **Add Publishing Point** dialog box, on the **Basic Settings** tab, specify publishing point options, and then click **OK**:



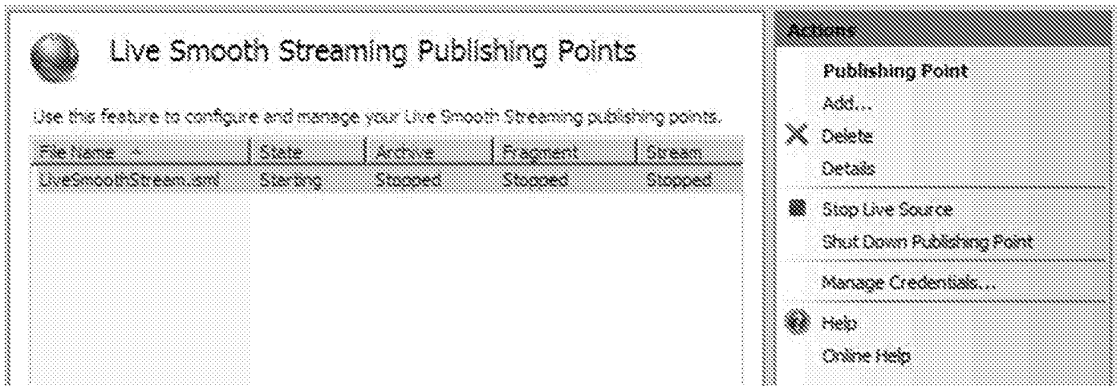
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0930
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- **File name**. The name of the publishing point definition file, which is a file with an .isml file name extension.
- **Title** (optional). A descriptive title for the publishing point.
- **Estimated duration**. If known, the duration of the live stream, formatted as <*days*.*hours*:*minutes*:*seconds*>, so that a client can scale its **Seek** bar to the length of the stream. For example, if you know that the live stream duration is 9 minutes, 57 seconds: **00:09:57**.
- **Live source type**. The **Push** source type means that live streams are posted to the publishing point from one or more live sources, either an encoder or a Live Smooth Streaming publishing point on another server. The **Pull** source type means that the publishing point gets the live streams by requesting them from a Live Smooth Streaming publishing point on another server.

4. In the **Live Smooth Streaming Publishing Points** feature page, click the publishing point that was just created, and then in the **Actions** pane, click **Start Publishing Point**.



5. The publishing point enters a **Starting** state.



The publishing point is now enabled. By default, it can broadcast live streams that are pushed to it by a live source, which can be either a Live Smooth Streaming encoder or a Live Smooth Streaming publishing point on another server, to Smooth Streaming-compatible clients. It is also

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0931
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

configured by default to archive the live streams that it receives as Smooth Streaming presentation files for later on-demand viewing.

With additional configuration, the publishing can distribute (or "push") live streams to Live Smooth Streaming publishing points on other servers. It also can be configured to get (or "pull") live streams from other Live Smooth Streaming publishing points. These latter two configuration options are very useful for distributing the live streams across a streaming server network in order to spread the load across all of the network computers.

For more information about additional settings that you can configure for each of these roles, see About publishing point roles.

The **Starting** state means that the publishing point is ready to receive live streams. When the live source connects to the publishing point and begins pushing content to it, the **Starting** state changes to **Started**, meaning that the publishing point is receiving the live streams. In addition, the **Stopped** status changes to **Started** for **Archive** (the publishing point is archiving the live streams), **Fragment**, (the publishing point is delivering the live streaming fragments to clients), and **Stream** (the publishing point is distributing the streams to other servers on the network), if these publishing point roles are enabled. For more information, see Monitoring publishing points.

You can disconnect the live source from this publishing point by clicking **Stop Live Source** in the **Actions** pane. For more information about managing connections, see Managing publishing points.

About publishing point roles

A publishing point can be configured to perform the following roles.

- **Client delivery**. Delivers the live streams from a live source to Smooth Streaming-compatible clients. You can configure the publishing point to source from an encoder or from a Live Smooth Streaming publishing point on another server in a distributed network.
- **Server distribution**. Distributes the live streams to Live Smooth Streaming publishing points on other servers (a push configuration) or gets the live streams from a Live Smooth Streaming publishing point on another server (a pull configuration) . You can configure Live Smooth Streaming publishing points on multiple servers to source from this publishing point, either by configuring this publishing point to "push' live streams to the servers or by configuring Live Smooth Streaming publishing points on the servers to "pull" live streams from this publishing point. This will balance the load across the servers in your Live Smooth Streaming network.
- **Archiving**. Archives the live streams from a live source (an encoder or Live Smooth Streaming publishing point on another server) to a Smooth Streaming presentation format for later on-demand viewing by Smooth Streaming-compatible clients.

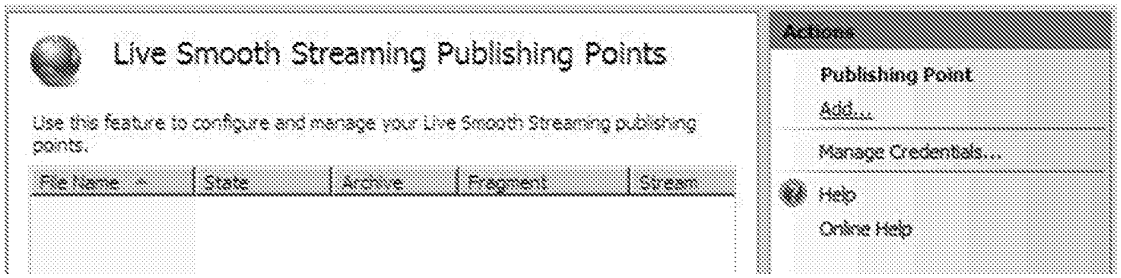
By default, the publishing point is configured to perform all of these roles. You can specify additional settings for each role, or disable one or more roles in the **Add Publishing Point** or **Edit Publishing Point**

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0932
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

dialog boxes. (You receive an error if you try to disable all three roles—at least one role must be enabled for a publishing point.)

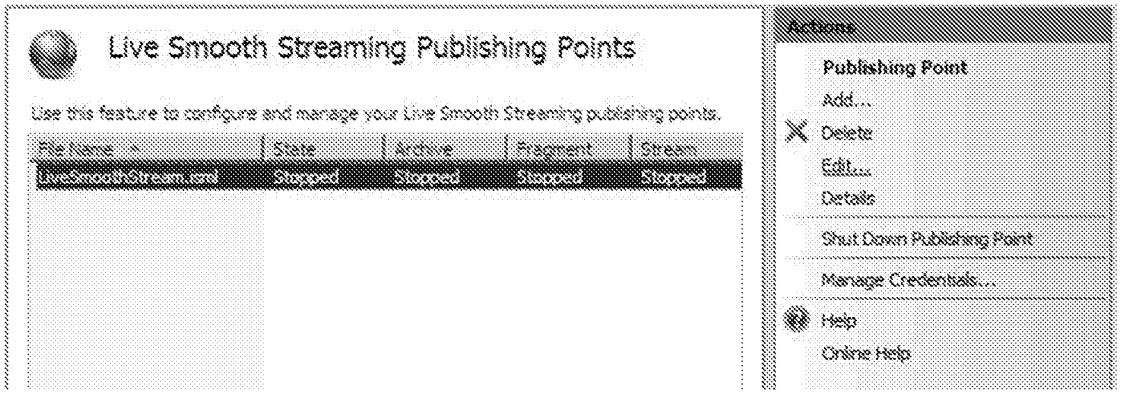
To access the additional role settings, do the following:

1. When creating a new publishing point, in the **Live Smooth Streaming Publishing Points** feature page, in the **Actions** pane, click **Add**.



-or-

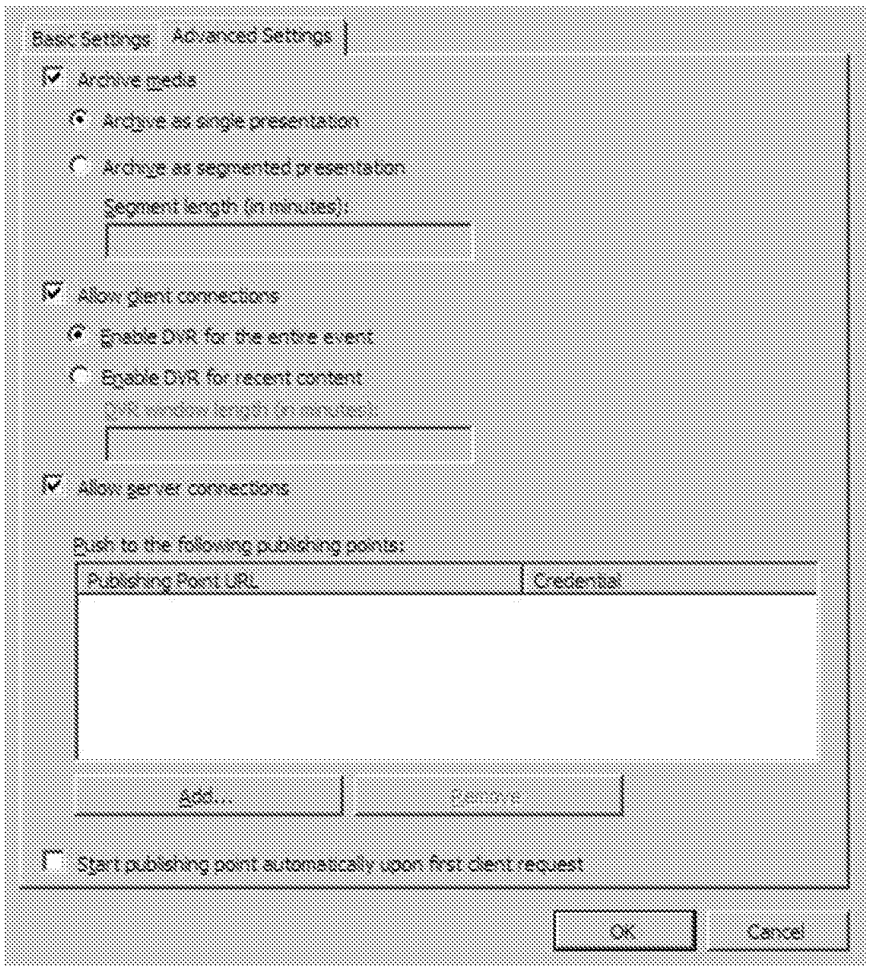
Click an existing publishing point in the **Live Smooth Streaming Publishing Points** feature page, and then in the **Actions** pane, click **Edit**.



Note that, as shown in the figure, a publishing point must be stopped before you can edit it. For more information, see Managing publishing points.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0933
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2. In the **Add Publishing Point** dialog box or **Edit Publishing Point** dialog box, click the **Advanced Settings** tab.



In the figure, the **Advanced Settings** tab shows the default settings for a publishing point. Note that Live Smooth Streaming publishing points on other servers can pull live streams from this publishing point because the **Allow server connections** option is selected. However, this publishing point cannot yet push live streams to Live Smooth Streaming publishing points on other servers because the URLs of the remote publishing points are not specified.

This section contains the following topics that describe how to configure the publishing point roles:

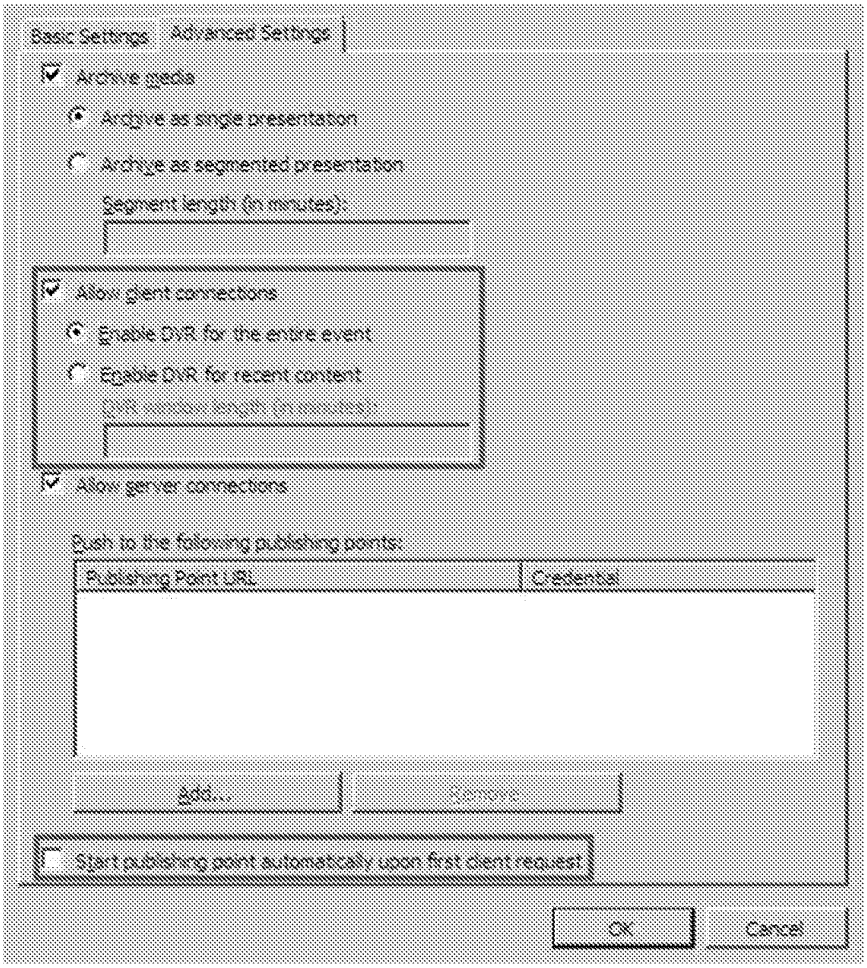
- Broadcasting live streams
- Distributing live streams
- Archiving live streams

For more information about how to configure publishing point roles, see Creating and Managing Publishing Points (http://go.microsoft.com/fwlink/?LinkId=166178).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0934
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Broadcasting live streams

To enable the publishing point to return live streams when requested by clients, configure the highlighted options:



- **Allow client connections**. Select this option to enable the publishing point to return live streams when requested by clients.

  If you choose this option, select one of the following options that specify how the clients are allowed to use their DVR transport (**Rewind**, **Fast-Forward**, and **Seek**) controls:

  - **Enable DVR for the entire live event**. Enables clients that join the live stream in-progress to rewind to the beginning of the live stream, and then fast-forward and seek to access any part of the live stream that has been published by the live source. This setting creates an archive of the entire live stream, which is discarded after the publishing point is shut down.
  - **Enable DVR for recent content**. Enables clients that join the live stream in-progress to rewind, fast-forward and seek within the part of the live stream that you specify in the **DVR**

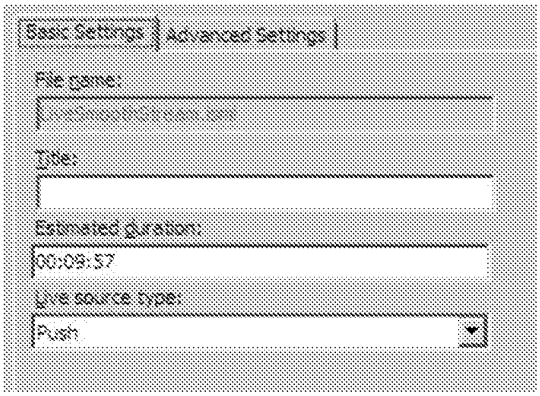
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0935
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

window length box. For example, if you enter **30** in the DVR window length box, clients can rewind 30 minutes of the live stream, and then fast-forward and seek to access any part of the live stream that has been published by the live source from that point forward. This setting creates an archive of the live stream with a duration that is equal to the **DVR window length**. As the live stream plays, archived content that is older than this setting is discarded.

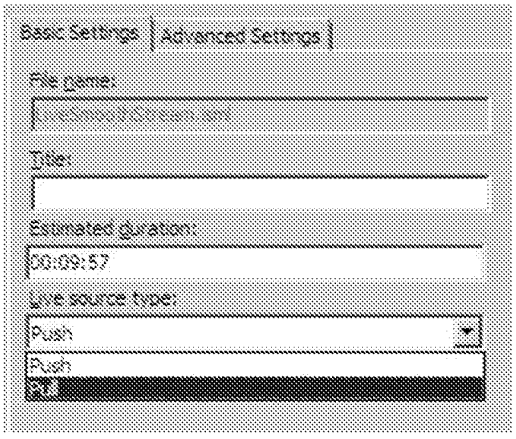
- **Start publishing point automatically upon first client request**. Select this option to configure the publishing point to start automatically when it receives a client request. If you select this option, you do not have to start the publishing point in the **Actions** pane. Clients must then wait for the live source to deliver content to the publishing point before they can view the stream.

Another way to source live streams for clients is to request (or "pull") them from a Live Smooth Streaming publishing point on another server. To do this, you must change the publishing point configuration from the default "Push" configuration to a "Pull" configuration. To do this, do the following:

1. Click the **Basic Settings** tab.

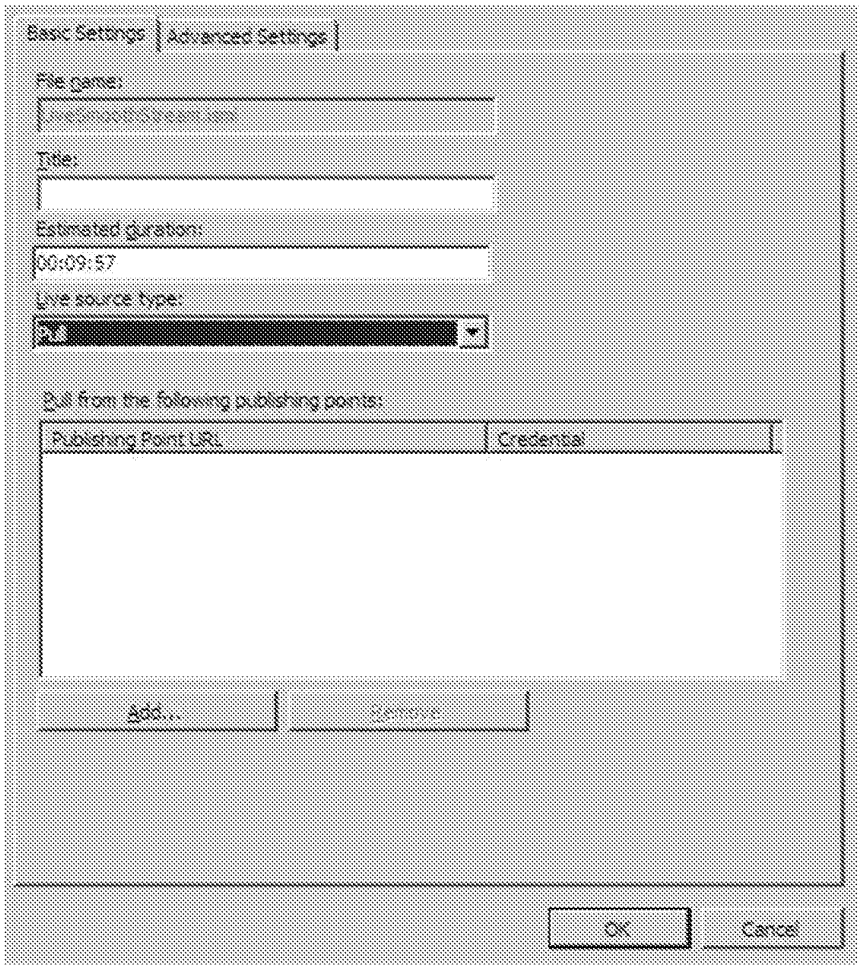


2. In **Live source type**, select **Pull**.

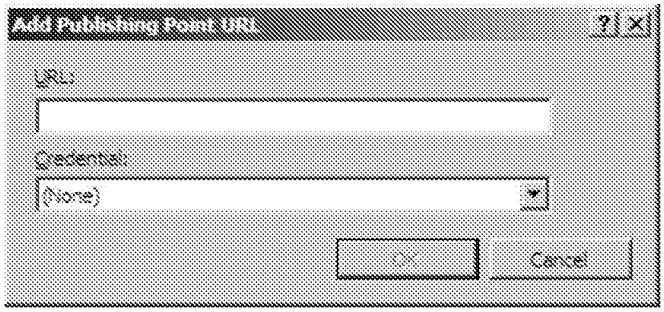


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0936
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3. Below the **Publishing Point URL** list that appears. Click **Add**.



4. Specify settings in the **Add Publishing Point URL** dialog box:



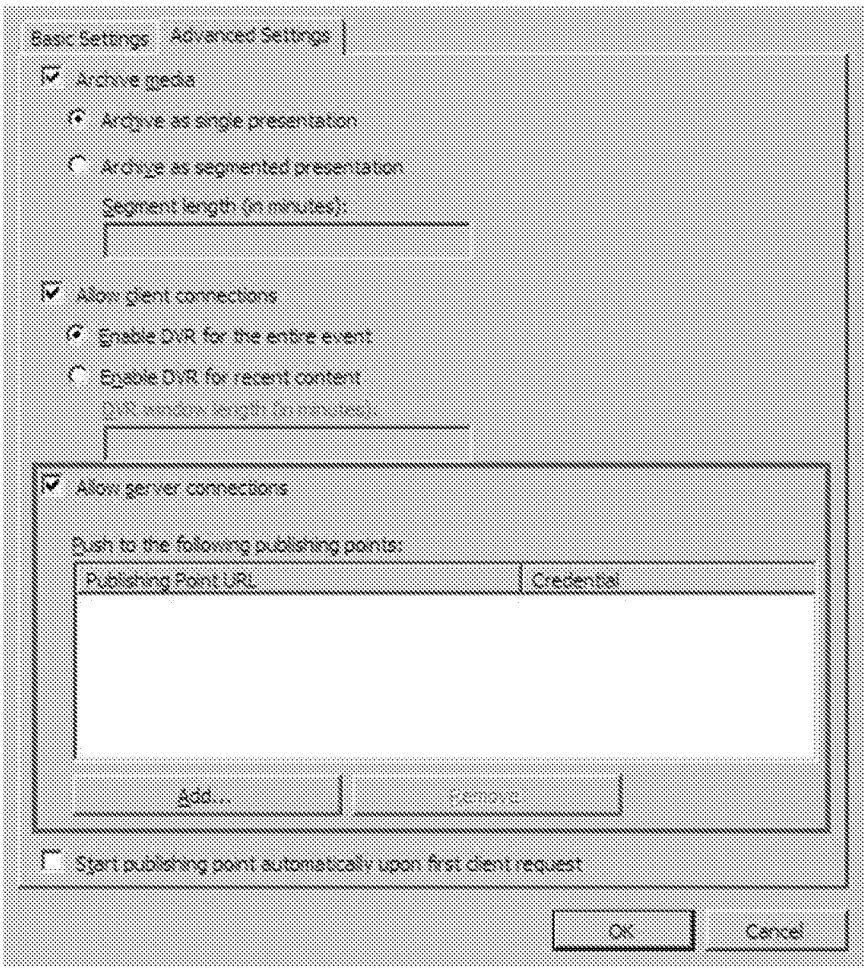
- **URL**. Enter the URL of the remote Live Smooth Streaming publishing point (for example, http://*ServerName*/*SiteName*/*PublishingPointName*.isml) from which this publishing point will pull live streams. The URL must begin with the http:// or https:// URL prefix and end with the .isml file name extension.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0937
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- **Credential**. If the connection to the publishing point requires authentication, select the friendly name of the credential with the appropriate user name and password. For more information, see Managing credentials.

Distributing live streams

To enable a publishing point to return live streams when they are requested by Live Smooth Streaming publishing points on other servers in a distributed network, configure the highlighted options:

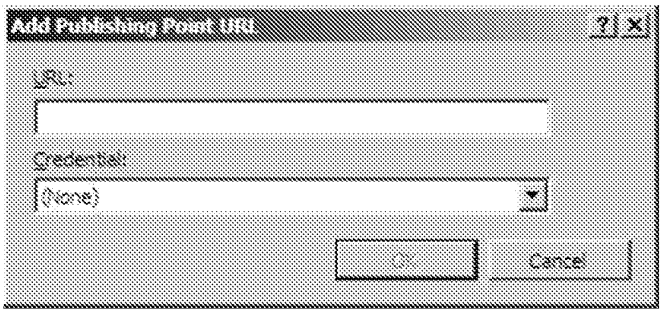


- **Allow server connections**. Select this option to enable the publishing point to push live streams to Live Smooth Streaming publishing points on other servers. The live streams can be pushed to multiple publishing points to balance the load across the servers in your network. To specify URLs of additional publishing points to which this publishing point will push live streams, do the following:

  1. Click **Add**.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0938
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2. Specify settings in the **Add Publishing Point URL** dialog box:

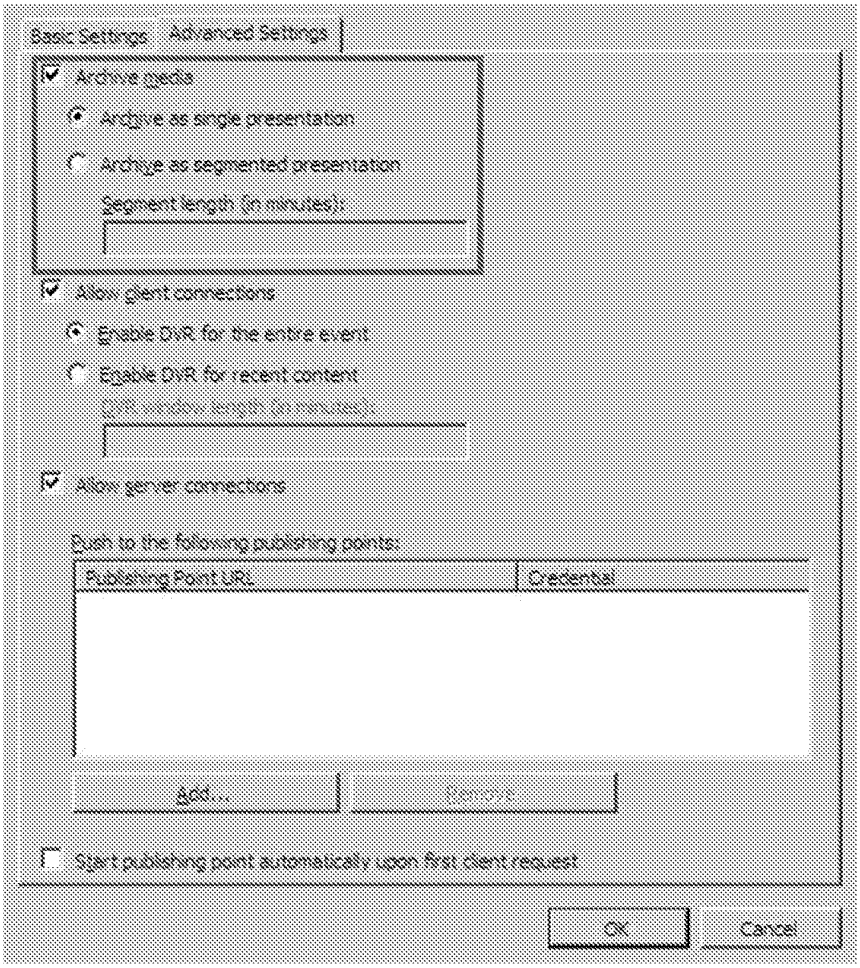


- **URL**. Enter the URL of the remote Live Smooth Streaming publishing point (for example, http://*ServerName*/*SiteName*/*PublishingPointName*.isml) to which this publishing point will push live streams. The URL must begin with the http:// or https:// URL prefix and end with the .isml file name extension.
- **Credential**. If the connection to the publishing point requires authentication, select the friendly name of the credential with the appropriate user name and password. For more information, see Managing credentials.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0939
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Archiving live streams

To enable a publishing point to archive live streams to Smooth Streaming presentation files for later on-demand viewing, configure the highlighted options:



- **Archive media**. Select this option to store the live streams as a Smooth Streaming presentation that can be served to clients for later on-demand viewing by using the Smooth Streaming extension for IIS 7. By default, the live streams are archived to a Smooth Streaming presentation folder in %*SystemDrive*%\inetpub\media\archives*SiteName**PublishingPointFileName*-isml. For more information about how to change the archive directory, see Configuring the archive path.

  If you choose to archive the live streams, select one of the following options that specify how the live streams are archived:

  - **Archive as single presentation**. Archives the live stream as a single Smooth Streaming presentation, which contains the following files: one contiguous .ismv file per bit rate, a presentation server manifest (.ism) file, and a client manifest (.ismc) file.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0940
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- **Archive as segmented presentation**. Archives the live stream as multiple Smooth Streaming presentations. The duration of each presentation is specified by entering a value (in minutes) in the **Segment length box**.

  Choose this option if you want to split a long-running broadcast into shorter segments for on-demand viewing. For example, you can archive a live stream that lasts 24 hours into 24 on-demand Smooth Streaming presentations, each lasting one hour by entering a value of **60**.

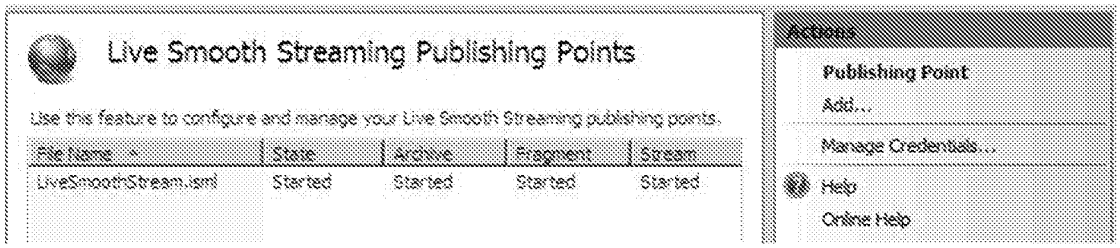
To make the archived live streams available as on-demand Smooth Streaming presentations, you must copy the archived presentation files to a Web site or directory on a Web server with the IIS Smooth Streaming extension installed. The IIS Smooth Streaming extension enables delivery of on-demand Smooth Streaming presentations. For more information, see Copying the Smooth Streaming presentation to the Web server.

Note

- Because the archived live stream does not include a Silverlight player template, you must add one to the directory to enable on-demand streaming to clients. For more information about adding a Smooth Streaming player template to the archive directory, see Playing Smooth Streams.

Monitoring publishing points

You can use the **Live Smooth Streaming Publishing Points** feature page to view the status of a publishing point's connection with a live source (which can be a Live Smooth Streaming encoder or a Live Smooth Streaming publishing point on another server) and its role configuration.



This section describes the status values that can be displayed.

State

The **State** column displays the status of the publishing point's connection to a live source, such as an encoder or another Live Smooth Streaming publishing point. The following status values can be displayed:

- **Idle**. The publishing point is not started and cannot receive live streams. To start it, select the publishing point and then click the **Start Publishing Point** link in the **Actions** pane.
- **Starting**. The publishing point is enabled and is waiting to receive live streams from the live source. Publishing points in this state cannot accept requests for live streams.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0941
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- **Started**. The publishing point is receiving live streams from the live source and can accept requests for live streams.
- **Stopping**. The publishing point is stopping because the live source is not available. This occurs if the Live Smooth Streaming presentation ended. In planned stoppages, the presentation has ended on-schedule or the server administrator clicked the **Stop Live Source** link in the **Actions** pane in IIS Manager. In unplanned stoppages, the presentation has ended due to an unforeseen event, such as a power outage or other error condition. This is a temporary condition that occurs just before the publishing point stops.
- **Stopped**. The publishing point is stopped. Note that a stopped publishing point can still deliver archived live streams if client delivery is enabled for the publishing point and client DVR settings are enabled for the publishing point. For more information about how to configure client delivery of live streams, see Broadcasting live streams.
- **Error**. The publishing point encountered an error and logged an event to Windows Event Viewer.
- **Shutting down**. The publishing point is being shutdown so that the publishing point cannot accept HTTP requests. After the publishing point is shutdown, the **State** status will return to **Idle**.

Archive

The **Archive** column displays the status of the live streams that are being archived. The following status values can be displayed:

- **Started**. The publishing point is archiving live streams.
- **Stopped**. Archiving of the live streams has stopped because the live source is not available. This occurs if the Live Smooth Streaming presentation ended or the connection between the live source and the publishing point was lost.
- **Disabled**. Archiving is not enabled for the publishing point. For more information about how to configure archiving of live streams, see Archiving live streams.

Fragment

The **Fragment** column displays the status of the live stream fragments that are being provided to clients. The following status values can be displayed:

- **Started**. The publishing point is receiving live streams from the live source and can accept client requests for live streams.
- **Stopped**. The publishing point is stopped because the live source is not available. This occurs if the Live Smooth Streaming presentation ended or the connection between the live source and the publishing point was lost. The publishing point can still deliver archived live streams if client DVR settings are enabled for the publishing point.
- **Disabled**. Client delivery is not enabled for the publishing point. For more information about how to configure client delivery of live streams, see Broadcasting live streams.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0942
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Stream

The **Stream** column displays the status of the live streams that are being distributed to Live Smooth Streaming publishing points on other servers. The following status values can be displayed:

- **Started**. The publishing point is receiving live streams from the live source and can accept server requests for live streams.
- **Stopped**. The publishing point is stopped because the live source is not available. This occurs if the Live Smooth Streaming presentation ended or the connection between the live source and the publishing point was lost. The publishing point cannot accept server requests for live streams.
- **Disabled**. Server distribution is not enabled for the publishing point. For more information about how to configure server distribution of live streams, see Distributing live streams.

Managing credentials

If you plan to distribute live streams to multiple Live Smooth Streaming publishing points in a network, it is a good idea to apply security measures to the connections between publishing points. This section describes how to create credentials that you can apply to a Live Smooth Steaming publishing point that you will use to pull live streams from a publishing point on another server (often called an *upstream publishing point*), in which access must be authorized.

Note

- For instructions on creating user accounts and access policies on the Web server that will be used to distribute live streams, and to review best practices for delivering live streams using enhanced security measures, see "Securing the Upstream Publishing Point" in Syndicating Content between Servers (http://go.microsoft.com/fwlink/?LinkId=166167).

This section contains the following topics:

- Configuring credentials
- Applying credentials

Configuring credentials

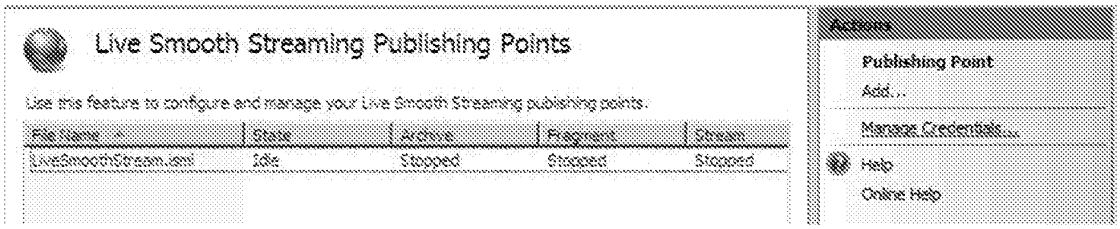
To create credentials for a Live Smooth Streaming publishing point that you will use to pull live streams from a publishing point on another server, do the following:

1. In the **Connections** pane, click the Web site or directory that contains the publishing point that you want to use for delivering live streams that are sourced from an upstream publishing point, and then in the **Home** page, double-click the **Live Smooth Streaming Publishing Points** icon.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0943
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

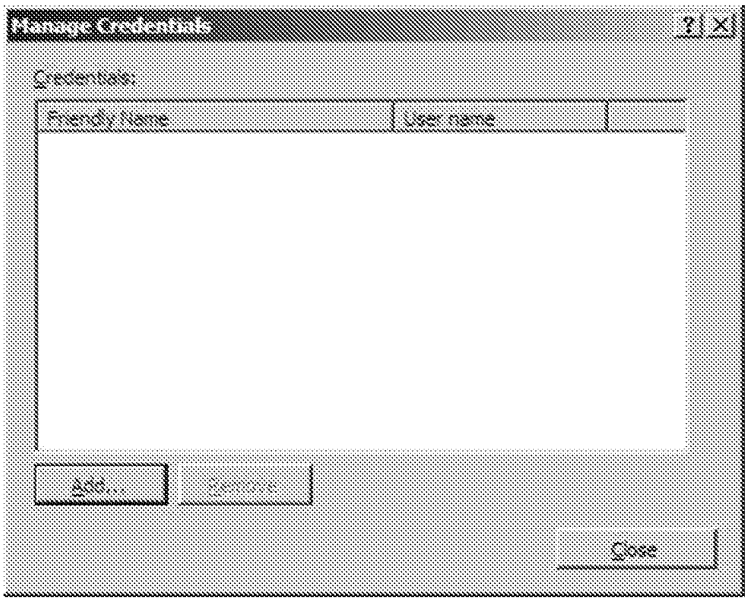


2. In the **Actions** pane, click **Manage Credentials**.

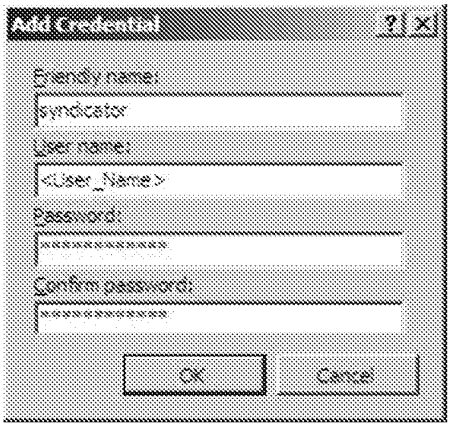


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0944
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3. In the **Manage Credentials** dialog box, click **Add**.



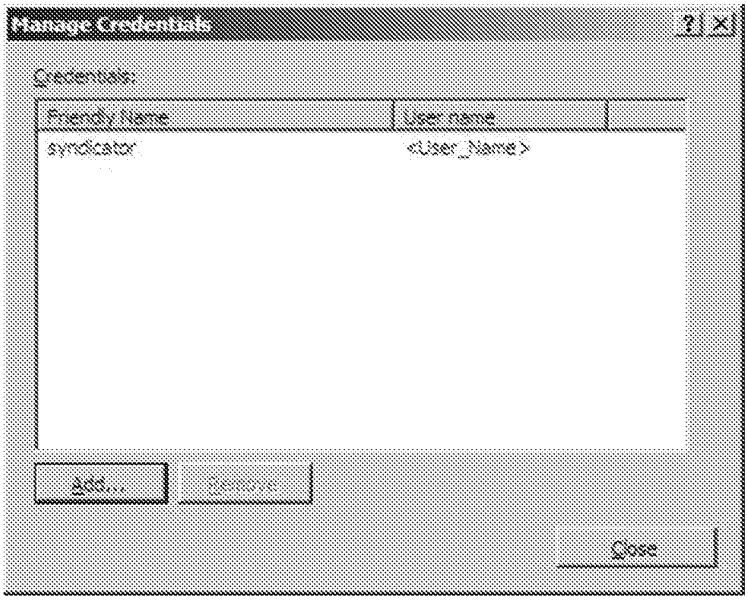
4. In the **Add Credential** dialog box, specify the following options, and then click **OK**:



- **Friendly name**. Type a descriptive friendly name for the credentials that will help you remember which upstream publishing point they will be used for. The friendly name is displayed in a **Credential** list in the **Add Publishing Point URL** dialog box for easy identification and selection.
- **User name**. Type the name of the user account that can connect to the upstream publishing point.
- **Password**. Type the password associated with the user account that can connect to the upstream publishing point.
- **Confirm password**. Retype the password associated with the user account that can connect to the upstream publishing point.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0945
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The credentials that you created for the upstream publishing point are now displayed in the **Manage Credentials** dialog box.



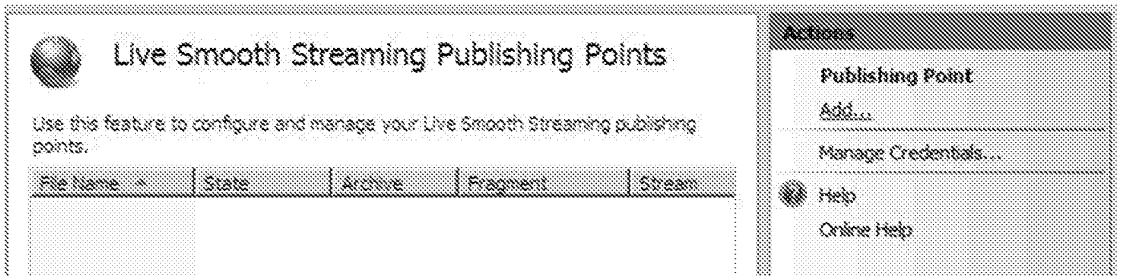
You can repeat the previous steps to create additional credentials for upstream Live Smooth Streaming publishing points that require authorization and they will be displayed in this dialog box.

To apply these credentials to the publishing point, see the following topic.

Applying credentials

To apply credentials to a Live Smooth Streaming publishing point that you will use to pull live streams from a publishing point on another server, do the following:

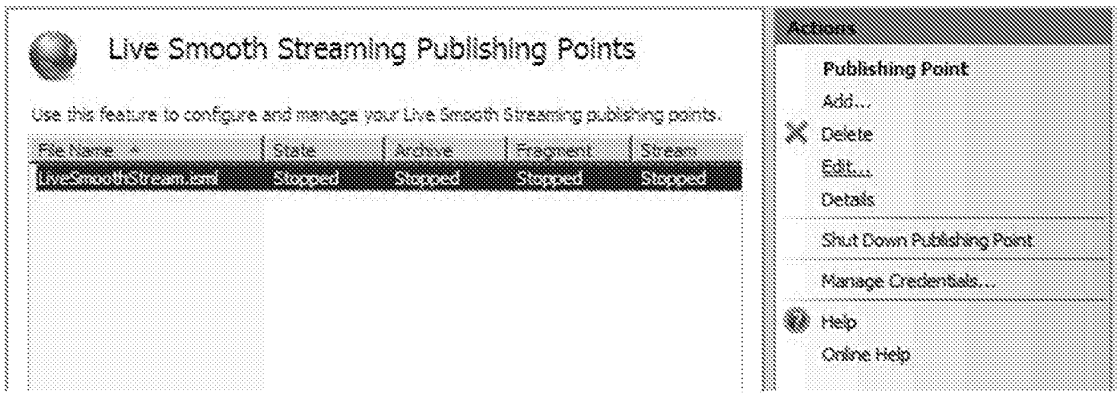
1. When creating a new publishing point, in the **Live Smooth Streaming Publishing Points** feature page, in the **Actions** pane, click **Add**.



-or-

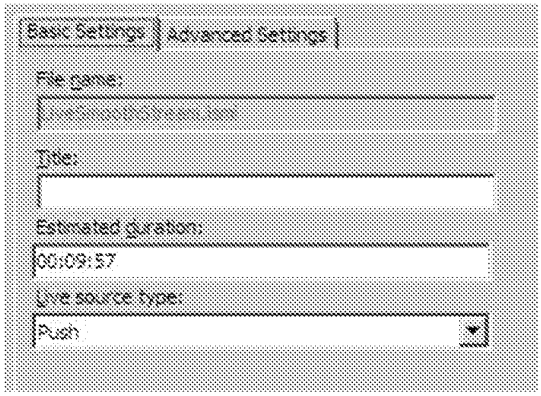
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0946
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Click an existing publishing point in the **Live Smooth Streaming Publishing Points** feature page, and then in the **Actions** pane, click **Edit**.



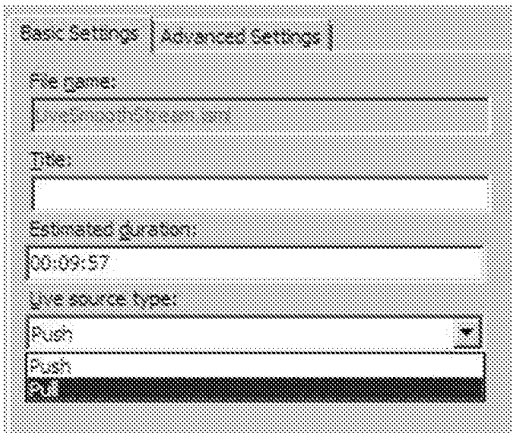
Note that, as shown in the figure, a publishing point must be stopped before you can edit it. For more information, see Managing publishing points.

2. In the **Add Publishing Point** dialog box or **Edit Publishing Point** dialog box, click the **Basic Settings** tab.

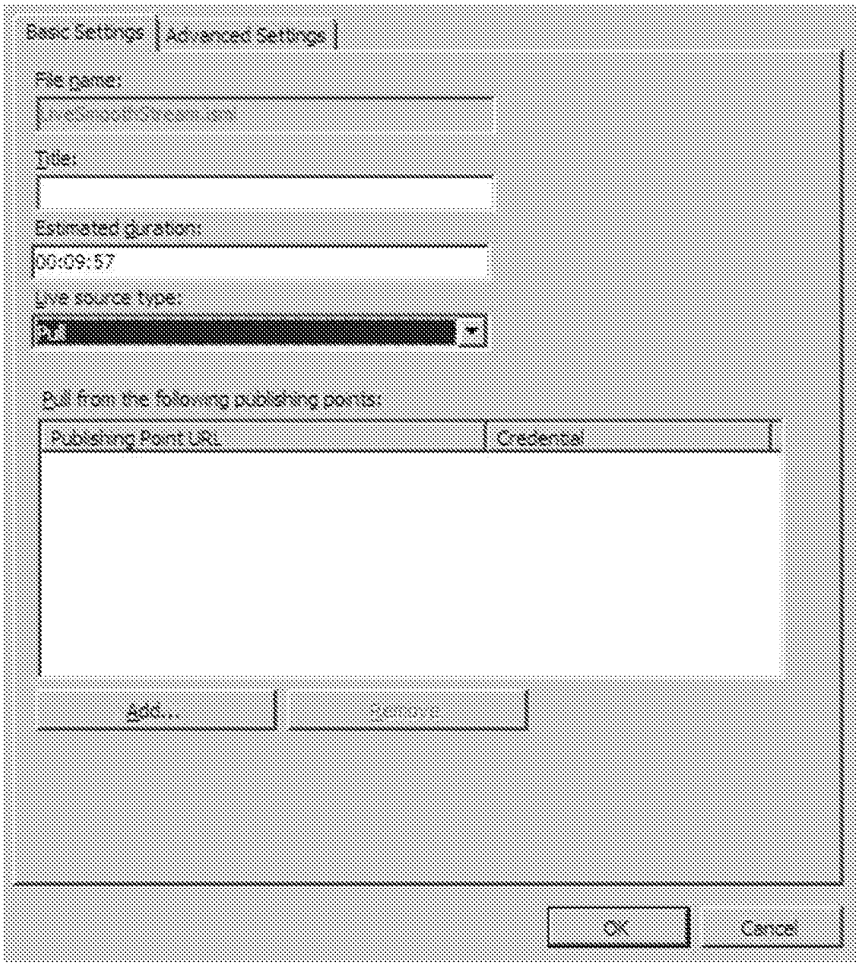


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0947
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3. In **Live source type**, select **Pull**.

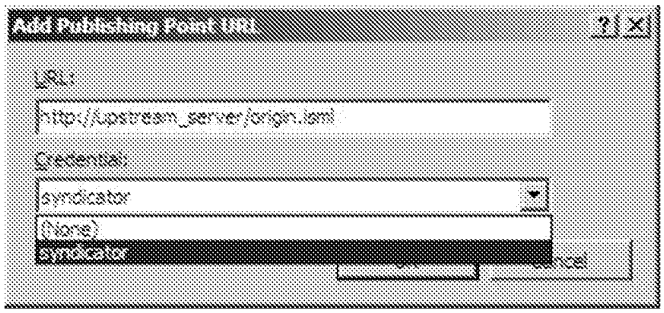


4. Below the **Publishing Point URL** list that appears. Click **Add**.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0948
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5. Specify settings in the **Add Publishing Point URL** dialog box:

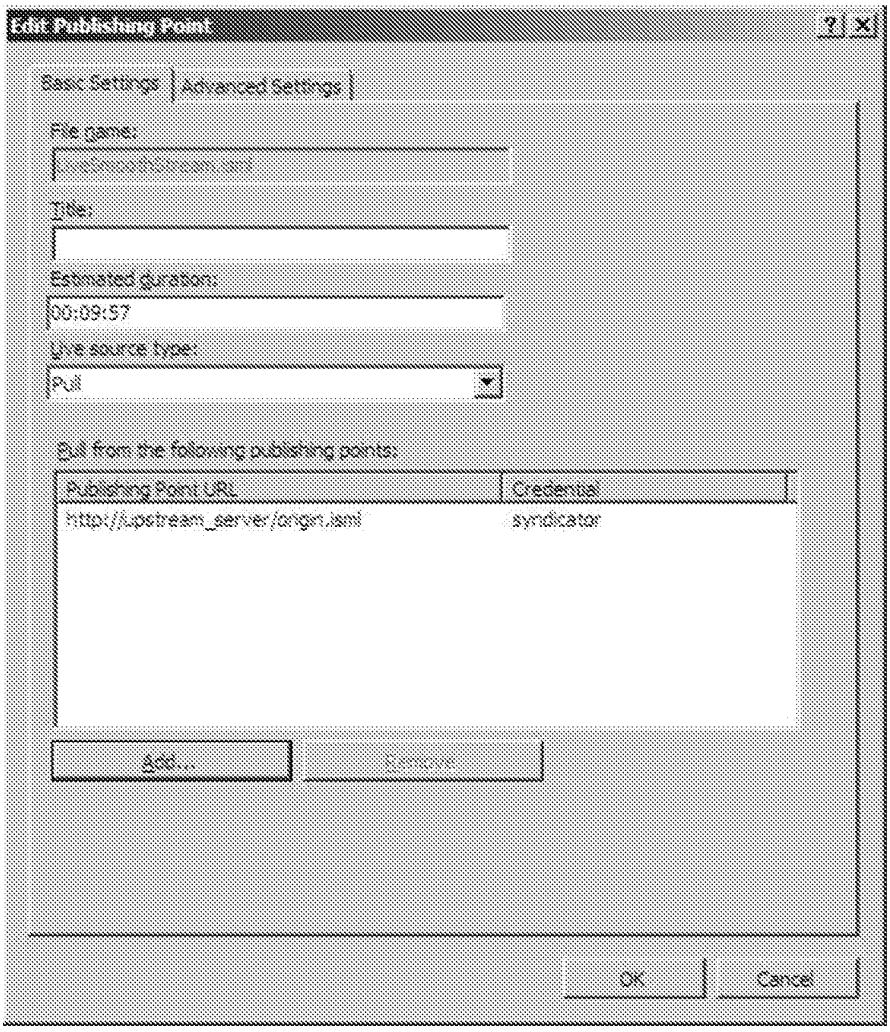


- **URL**. Enter the URL of the upstream Live Smooth Streaming publishing point from which this publishing point will pull live streams. The URL must begin with the http:// or https:// URL prefix and end with the .isml file name extension.
- **Credential**. In the dropdown list, select the friendly name of the credential with the appropriate user name and password.

The following figure shows that this publishing point (LiveSmoothStream.isml) will source live streams from the upstream publishing point URL **http://upstream_server/origin.isml**, using the credentials

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0949
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

associated with the **syndicator** friendly name.



The upstream publishing point must be configured to distribute live streams to this publishing point (the *downstream publishing point*). For more information, see Distributing live streams.

Configuring advanced settings

Use the **Live Smooth Streaming** feature page to configure caching and archiving options for Live Smooth Streaming. The settings that you make on this page apply to all of the Live Smooth Streaming publishing points on the Web server.

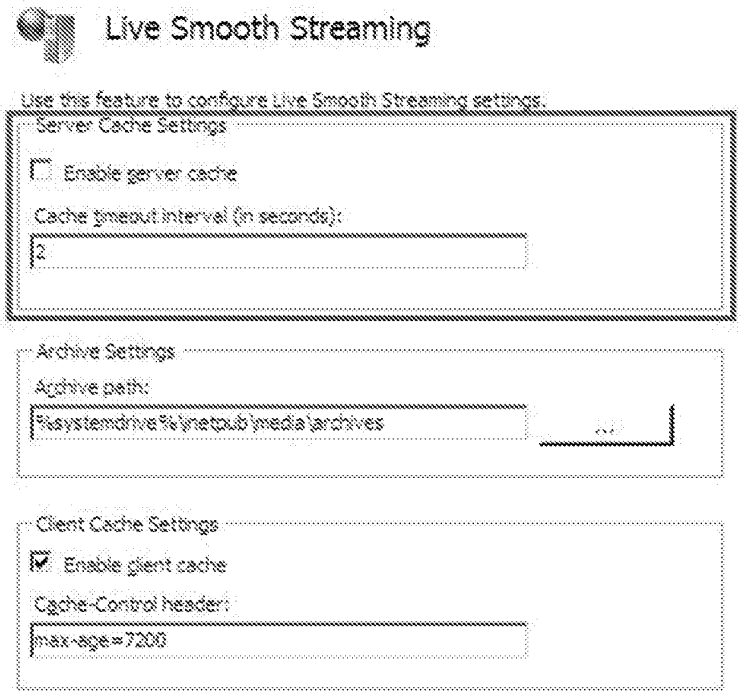
This section contains the following topics:

- Configuring server cache settings
- Configuring the archive path
- Configuring client cache settings

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0950
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Configuring server cache settings

To configure output caching to improve Web server performance and reduce latency in the HTTP response to clients, configure the highlighted settings:

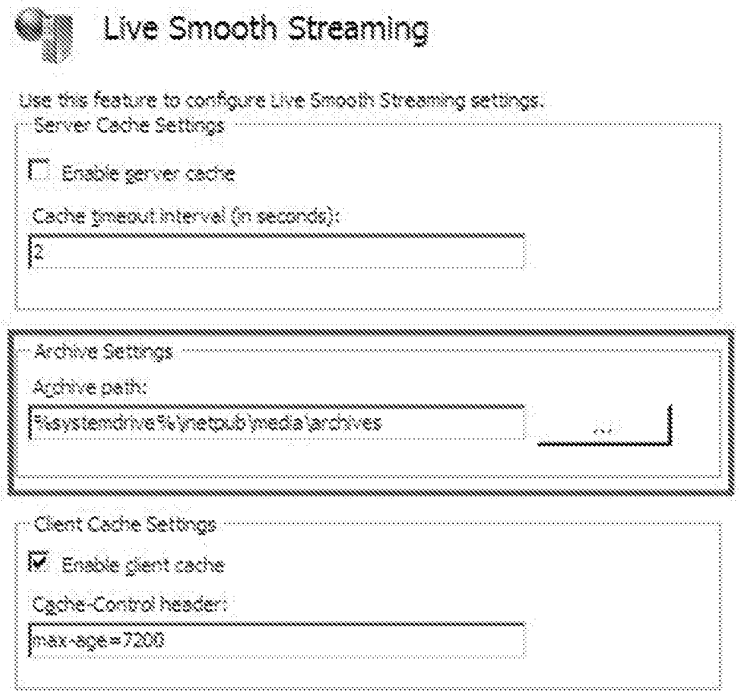


- **Enable server cache**. Uses the **Output Caching** feature settings in IIS Manager to cache live smooth stream metadata, the first time the stream is requested by a client. This reduces latency in the HTTP response to subsequent clients that request the same live stream. For more information about how to configure output caching, see Configuring Output Caching in IIS 7 (http://go.microsoft.com/fwlink/?LinkId=166164).
- **Cache timeout interval**. Enter a time, in seconds, that specifies how long the live smooth stream metadata is cached.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0951
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Configuring the archive path

To specify the root directory where live streams are saved as a Smooth Streaming presentation for later on-demand viewing, configure the highlighted settings.



- **Archive path**. By default, the Live Smooth Streaming feature archives live streams to %*SystemDrive*%\inetpub\media\archives for publishing points that have archiving enabled. To archive live streams to a different directory on the local computer, enter a physical path, such as C:\Archive. To archive live streams to a remote share, enter the Universal Naming Convention (UNC) path, such as *ServerName**ShareName*. Optionally, click the **Browse** (**…**) button to locate and select a directory folder on the local computer or remote share. The IIS_IUSRS group on the Web server must have Read and Write permissions to the new directory.

Note

- You must enable the **Archive media** option for a publishing point so that it can archive live streams to the new directory. For more information, see Archiving live streams.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0952
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Configuring client cache settings

To control how client browsers and proxy servers cache the live smooth streaming content returned by the Web server, configure the highlighted settings.



- **Enable client cache**. Enables caching of live smooth streaming content by clients, according to caching rules specified in **Cache-Control header**.
- **Cache-Control header**. Enter a Cache-Control header directive to be included in the HTTP header response sent to clients. The Cache-Control header directive specifies the conditions and restrictions for caching live smooth streaming content on the client. For a list of valid directives that you can enter, see Cache-Control (http://go.microsoft.com/fwlink/?LinkId=166165).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0953
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Playing Smooth Streams

A Smooth Streaming-compatible Silverlight player template must be installed on the Smooth Streaming-enabled Web server so that Silverlight-based clients can play Smooth Streams. Clients should have the latest version of Microsoft Silverlight (http://go.microsoft.com/fwlink/?LinkId=166168) installed for the best playback experience.

Typically, a Silverlight player template consists of the following files:

- ***.html** or ***.htm**. The template home page, which client Web browsers with a Silverlight plug-in installed use to access the stream. Typically, this file is named **default.html**.
- ***.xap**. Silverlight player template files, which specify the skin (the visual appearance) that surrounds the video and the playback elements in the skin design.

Encoders that create on-demand Smooth Streaming output often include Silverlight player template files with the encoded Smooth Streaming presentation files. To determine if a Smooth Streaming presentation on your Web server contains the template files, see Viewing Smooth Streaming presentation details.

If no template files are present, you can either use a prepackaged player template that is included with Microsoft Silverlight Media Framework 2.0 (http://go.microsoft.com/fwlink/?LinkID=194570)or create your own player by using IIS Smooth Streaming Client (http://go.microsoft.com/fwlink/?LinkID=194571) APIs.

Encoders that create live Smooth Streaming output do not include a Silverlight player template and so you must either download Smooth Streaming Silverlight player template files or create your own player.

This section contains the following topics that describe how to use the prepackaged player template that is provided by the Silverlight Media Framework:

- Installing the sample Smooth Streaming player
- Enabling playback of Smooth Streams

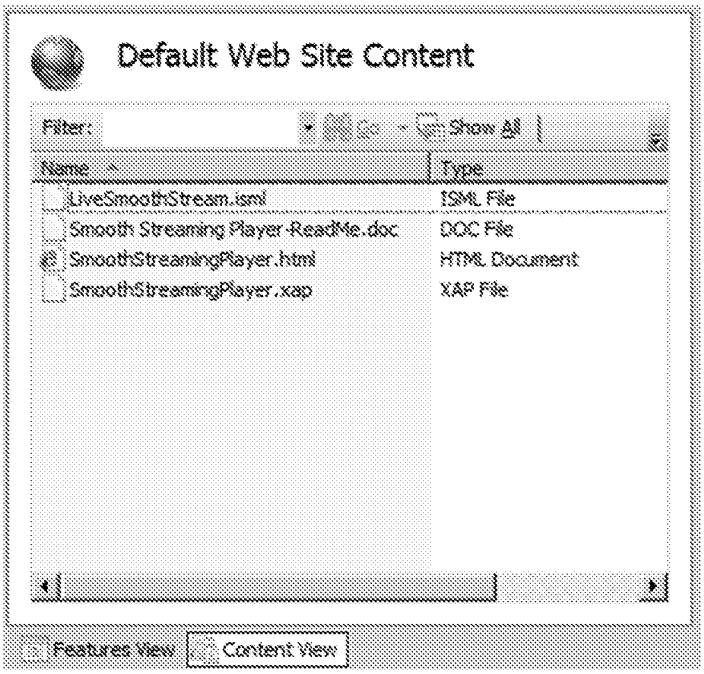
Installing the sample Smooth Streaming player

To install the Silverlight Media Framework 2.0 SmoothStreamingPlayer.XAP, do the following.

1. Download the Microsoft SMFv2 - Smooth Streaming HTML template (http://go.microsoft.com/fwlink/?LinkId=199942) from Microsoft CodePlex to the Smooth Streaming Web server.
2. Extract the files to the Web site or directory that you want clients to access when viewing Smooth Streams. For example, the following figure shows files that are extracted to the **Default Web Site**, which also contains a Live Smooth Streaming publishing point definition file

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0954
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

(LiveSmoothStream.isml):



The following files are included in the download:

- **Smooth Streaming Player-ReadMe.doc**. Detailed configuration instructions.
- **SmoothStreamingPlayer.html**. A sample Web page used by clients to view Smooth Streams. In the Web page, you must update the media reference to your on-demand presentation or live broadcast.
- **SmoothStreamingPlayer.xap**. A sample Silverlight player template that is optimized for Smooth Streaming.

Enabling playback of Smooth Streams

1. In Notepad, open <*DirectoryPath*>\SmoothStreamingPlayer.html.
2. In the **param** tag named **InitParams**, substitute the fully qualified URL of your Smooth Streaming presentation in place of the sample **mediaurl** property value, making sure to append **/Manifest** to the end of the URL.

   For on-demand Smooth Streams, specify the URL of the Smooth Streaming presentation server manifest file (.ism), for example:

   ```
   <param name="InitParams"
   value="mediaurl=http://iismedia7/BigBuckBunny.ism/Manifest" />
   ```

   For live Smooth Streams, specify the URL of the Live Smooth Streaming publishing point (.isml) that receives the encoded streams, for example:

   ```
   <param name="InitParams"
   ```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0955
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

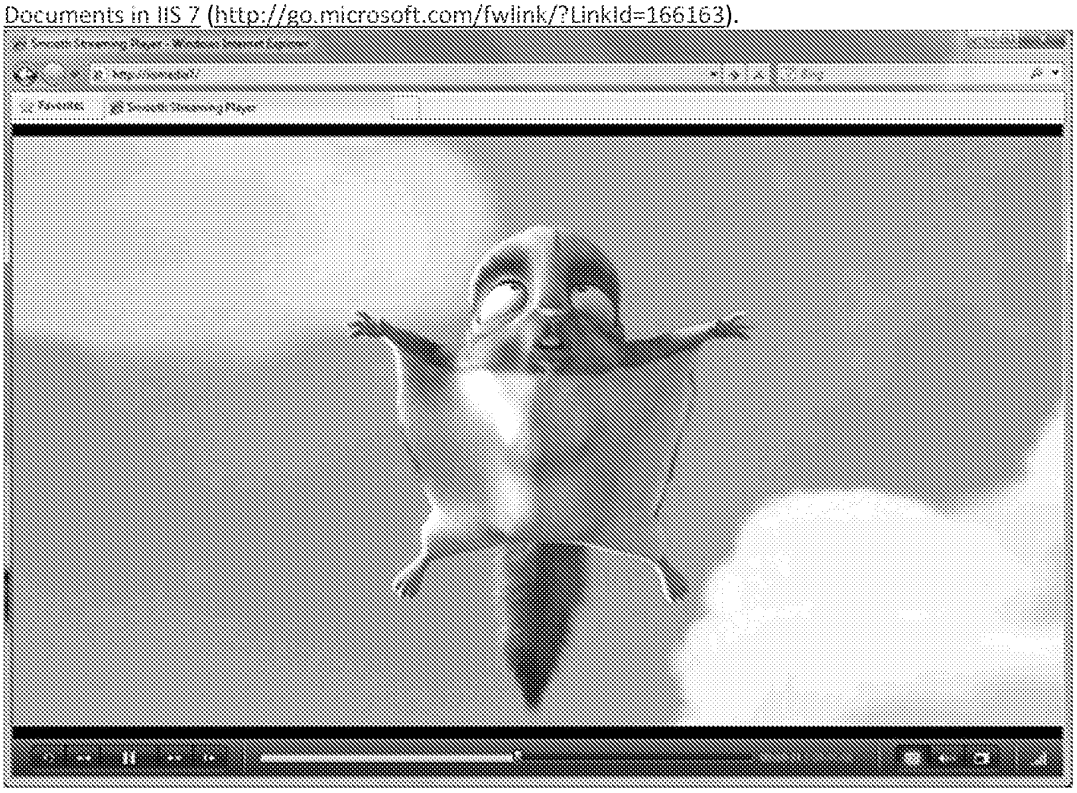
```
value="mediaurl=http://iismedia7/LiveSmoothStream.isml/Manifest"
/>
```

The **mediaurl** property is the only property required for clients to begin playback. You can add additional properties to **InitParams**. For complete instructions, see the Readme file (Smooth Streaming Player-ReadMe.doc) that is included in the download.

3. The Web page can now be used to give clients access to your Smooth Stream (http://*ServerName*/*DirectoryName*/SmoothStreamingPlayer.html).

   -or-

   If you specify that **default.html** be returned when clients do not specify a specific file in the directory, you can rename the file to **default.html** and then use the following URL: **http://*ServerName*/*DirectoryName*/**. For more information, see Configuring Default Documents in IIS 7 (http://go.microsoft.com/fwlink/?LinkId=166163).



Note

- To embed the player into another Web page, you can copy the contents of the **object** tag from SmoothStreamingPlayer.html and paste it into the Web page. If necessary, update the **width** and **height** property values for your Web site design. For complete instructions, see the Readme file (Smooth Streaming Player-ReadMe.doc) that is included in the download.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0956
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Additional Resources

- Smooth Streaming Technical Overview (http://go.microsoft.com/fwlink/?LinkId=166177). Provides background information and technical details about IIS Smooth Streaming and its benefits.
- IIS Media Services (http://go.microsoft.com/fwlink/?LinkId=166171). Provides technical articles, blogs, and a forum to help you learn more about and use the features that comprise the integrated Web-based media delivery platform, including the Smooth Streaming extensions for IIS 7.
- Expression Encoder 4 (http://go.microsoft.com/fwlink/?LinkId=165795). Describes how to use Expression Encoder to bring your edited video projects to the Web for high-quality playback in Microsoft Silverlight scenarios.
- Microsoft Silverlight (http://go.microsoft.com/fwlink/?LinkID=165789). If you are a designer or developer, this site provides a broad range of information, samples, and tools to help you with your Silverlight player development.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0957
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Legal Notice

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2010 Microsoft Corporation. All rights reserved.

Microsoft, Expression, the Expression logo, Silverlight, the Silverlight logo, Windows, the Windows logo, Windows Server, Windows Vista, PlayReady, and DirectX are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Big Buck Bunny images © copyright Blender Foundation | www.blender.org/

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0958
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



Arithmetic Coding + Statistical Modeling = Data Compression

Part 1 - Arithmetic Coding

by Mark Nelson

Dr. Dobb's Journal February, 1991

This page contains my original text and figures for the article that appeared in the February, 1991 DDJ. The article was originally written to be a 2 part feature, but was cut down to 1 part. Links to Part 2 are here and at the end of the article.

Arithmetic Coding + Statistical Modeling = Data Compression

Part 1 - Arithmetic Coding

by Mark Nelson

Most of the data compression methods in common use today fall into one of two camps: dictionary based schemes and statistical methods. In the world of small systems, dictionary based data compression techniques seem to be more popular at this time. However, by combining arithmetic coding with powerful modeling techniques, statistical methods for data compression can actually achieve better performance. This two-part article discusses how to combine arithmetic coding with several different modeling methods to achieve some impressive compression ratios. The first part of the article details how arithmetic coding works. The second shows how to develop some effective models that can use an arithmetic coder to generate high performance compression programs.

Terms of Endearment

Data compression operates in general by taking "symbols" from an input "text", processing them, and writing "codes" to a compressed file. For the purposes of this article, symbols are usually bytes, but they could just as easily be pixels, 80 bit floating point numbers, or EBCDIC characters. To be effective, a data compression scheme needs to be able to transform the compressed file back into an identical copy of the input text. Needless to say, it also helps if the compressed file is smaller than the input text!

Dictionary based compression systems operate by replacing groups of symbols in the input text with fixed length codes. A well-known example of a dictionary technique is LZW data compression. (See "LZW Data Compression" in the 10/89 issue of DDJ). LZW operates by replacing strings of essentially unlimited length with codes that usually range in size from 9 to 16 bits.

Statistical methods of data compression take a completely different approach. They operate by encoding symbols one at a time. The symbols are encoded into variable length output codes. The length of the output code varies based on the probability or frequency of the symbol. Low probability symbols are encoded using many bits, and high probability symbols are encoded using fewer bits.

In practice, the dividing line between statistical and dictionary methods is not always so distinct. Some

schemes can't be clearly put in one camp or the other, and there are always hybrids which use features from both techniques. However, the methods discussed in this article use arithmetic coding to implement purely statistical compression schemes.

Huffman Coding: The Retired Champion

It isn't enough to just be able to accurately calculate the probability of characters in a datastream, we also need a coding method that effectively takes advantage of that knowledge. Probably the best known coding method based on probability statistics is Huffman coding. D. A. Huffman published a paper in 1952 describing a method of creating a code table for a set of symbols given their probabilities. The Huffman code table was guaranteed to produce the lowest possible output bit count possible for the input stream of symbols, when using fixed length codes. Huffman called these "minimum redundancy codes", but the scheme is now universally called Huffman coding. Other fixed length coding systems, such as Shannon-Fano codes, were shown by Huffman to be non-optimal.

Huffman coding assigns an output code to each symbol, with the output codes being as short as 1 bit, or considerably longer than the input symbols, strictly depending on their probabilities. The optimal number of bits to be used for each symbol is the log base 2 of (1/p), where p is the probability of a given character. Thus, if the probability of a character is 1/256, such as would be found in a random byte stream, the optimal number of bits per character is log base 2 of 256, or 8. If the probability goes up to 1/2, the optimum number of bits needed to code the character would go down to 1.

The problem with this scheme lies in the fact that Huffman codes have to be an integral number of bits long. For example, if the probability of a character is 1/3, the optimum number of bits to code that character is around 1.6. The Huffman coding scheme has to assign either 1 or 2 bits to the code, and either choice leads to a longer compressed message than is theoretically possible.

This non-optimal coding becomes a noticeable problem when the probability of a character becomes very high. If a statistical method can be developed that can assign a 90% probability to a given character, the optimal code size would be 0.15 bits. The Huffman coding system would probably assign a 1 bit code to the symbol, which is 6 times longer than is necessary.

Adaptive Schemes

A second problem with Huffman coding comes about when trying to do adaptive data compression. When doing non-adaptive data compression, the compression program makes a single pass over the data to collect statistics. The data is then encoded using the statistics, which are unchanging throughout the encoding process.

In order for the decoder to decode the compressed data stream, it first needs a copy of the statistics. The encoder will typically prepend a statistics table to the compressed message, so the decoder can read it in before starting. This obviously adds a certain amount of excess baggage to the message.

When using very simple statistical models to compress the data, the encoding table tends to be rather small. For example, a simple frequency count of each character could be stored in as little as 256 bytes with fairly high accuracy. This wouldn't add significant length to any except the very smallest messages. However, in order to get better compression ratios, the statistical models by necessity have to grow in size. If the statistics for the message become too large, any improvement in compression ratios will be wiped out by added length needed to prepend them to the encoded message.

In order to bypass this problem, adaptive data compression is called for. In adaptive data compression, both the encoder and decoder start with their statistical model in the same state. Each of them process a single character at a time, and update their models after the character is read in. This is very similar to the way most dictionary based schemes such as LZW coding work. A slight amount of efficiency is lost by beginning the message with a non-optimal model, but it is usually more than made up for by not having to pass any statistics with the message.

The problem with combining adaptive modeling with Huffman coding is that rebuilding the Huffman tree is a very expensive process. For an adaptive scheme to be efficient, it is necessary to adjust the Huffman tree after every character. Algorithms to perform adaptive Huffman coding were not published until over 20 years after Huffman coding was first developed. Even now, the best adaptive Huffman coding algorithms are still relatively time and memory consuming.

Arithmetic Coding: how it works

It has only been in the last ten years that a respectable candidate to replace Huffman coding has been successfully demonstrated: Arithmetic coding. Arithmetic coding completely bypasses the idea of replacing an input symbol with a specific code. Instead, it takes a stream of input symbols and replaces it with a single floating point output number. The longer (and more complex) the message, the more bits are needed in the output number. It was not until recently that practical methods were found to implement this on computers with fixed sized registers.

The output from an arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols that went into its construction. In order to construct the output number, the symbols being encoded have to have a set probabilities assigned to them. For example, if I was going to encode the random message "BILL GATES", I would have a probability distribution that looks like this:

Character	Probability
SPACE	1/10
A	1/10
B	1/10
E	1/10
G	1/10
I	1/10
L	2/10
S	1/10
T	1/10

Once the character probabilities are known, the individual symbols need to be assigned a range along a "probability line", which is nominally 0 to 1. It doesn't matter which characters are assigned which segment of the range, as long as it is done in the same manner by both the encoder and the decoder. The nine character symbol set use here would look like this:

Character	Probability	Range
SPACE	1/10	0.00 - 0.10
A	1/10	0.10 - 0.20
B	1/10	0.20 - 0.30
E	1/10	0.30 - 0.40

G	1/10	0.40 - 0.50
I	1/10	0.50 - 0.60
L	2/10	0.60 - 0.80
S	1/10	0.80 - 0.90
T	1/10	0.90 - 1.00

Each character is assigned the portion of the 0-1 range that corresponds to its probability of appearance. Note also that the character "owns" everything up to, but not including the higher number. So the letter 'T' in fact has the range 0.90 - 0.9999....

The most significant portion of an arithmetic coded message belongs to the first symbol to be encoded. When encoding the message "BILL GATES", the first symbol is "B". In order for the first character to be decoded properly, the final coded message has to be a number greater than or equal to 0.20 and less than 0.30. What we do to encode this number is keep track of the range that this number could fall in. So after the first character is encoded, the low end for this range is 0.20 and the high end of the range is 0.30.

After the first character is encoded, we know that our range for our output number is now bounded by the low number and the high number. What happens during the rest of the encoding process is that each new symbol to be encoded will further restrict the possible range of the output number. The next character to be encoded, 'I', owns the range 0.50 through 0.60. If it was the first number in our message, we would set our low and high range values directly to those values. But 'I' is the second character. So what we do instead is say that 'I' owns the range that corresponds to 0.50-0.60 in the new subrange of 0.2 - 0.3. This means that the new encoded number will have to fall somewhere in the 50th to 60th percentile of the currently established range. Applying this logic will further restrict our number to the range 0.25 to 0.26.

The algorithm to accomplish this for a message of any length is is shown below:

```
Set low to 0.0
Set high to 1.0
While there are still input symbols do
    get an input symbol
    code_range = high - low.
    high = low + range*high_range(symbol)
    low = low + range*low_range(symbol)
End of While
output low
```

Following this process through to its natural conclusion with our chosen message looks like this:

New Character	Low value	High Value
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPACE	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0962
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

So the final low value, 0.2572167752 will uniquely encode the message "BILL GATES" using our present encoding scheme.

Given this encoding scheme, it is relatively easy to see how the decoding process will operate. We find the first symbol in the message by seeing which symbol owns the code space that our encoded message falls in. Since the number 0.2572167752 falls between 0.2 and 0.3, we know that the first character must be "B". We then need to remove the "B" from the encoded number. Since we know the low and high ranges of B, we can remove their effects by reversing the process that put them in. First, we subtract the low value of B from the number, giving 0.0572167752. Then we divide by the range of B, which is 0.1. This gives a value of 0.572167752. We can then calculate where that lands, which is in the range of the next letter, "I".

The algorithm for decoding the incoming number looks like this:

```
get encoded number
Do
    find symbol whose range straddles the encoded number
    output the symbol
    range = symbol low value - symbol high value
    subtract symbol low value from encoded number
    divide encoded number by range
until no more symbols
```

Note that I have conveniently ignored the problem of how to decide when there are no more symbols left to decode. This can be handled by either encoding a special EOF symbol, or carrying the stream length along with the encoded message.

The decoding algorithm for the "BILL GATES" message will proceed something like this:

Encoded Number	Output Symbol	Low	High	Range
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	SPACE	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.2	0.3	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

In summary, the encoding process is simply one of narrowing the range of possible numbers with every new symbol. The new range is proportional to the predefined probability attached to that symbol.

Decoding is the inverse procedure, where the range is expanded in proportion to the probability of each symbol as it is extracted.

Practical Matters

The process of encoding and decoding a stream of symbols using arithmetic coding is not too complicated. But at first glance, it seems completely impractical. Most computers support floating point numbers of up to 80 bits or so. Does this mean you have to start over every time you finish encoding 10 or 15 symbols? Do you need a floating point processor? Can machines with different floating point formats communicate using arithmetic coding?

As it turns out, arithmetic coding is best accomplished using standard 16 bit and 32 bit integer math. No floating point math is required, nor would it help to use it. What is used instead is a an incremental transmission scheme, where fixed size integer state variables receive new bits in at the low end and shift them out the high end, forming a single number that can be as many bits long as are available on the computer's storage medium.

In the previous section, I showed how the algorithm works by keeping track of a high and low number that bracket the range of the possible output number. When the algorithm first starts up, the low number is set to 0.0, and the high number is set to 1.0. The first simplification made to work with integer math is to change the 1.0 to 0.999...., or .111... in binary.

In order to store these numbers in integer registers, we first justify them so the implied decimal point is on the left hand side of the word. Then we load as much of the initial high and low values as will fit into the word size we are working with. My implementation uses 16 bit unsigned math, so the initial value of high is 0xFFFF, and low is 0. We know that the high value continues with FFs forever, and low continues with 0s forever, so we can shift those extra bits in with impunity when they are needed.

If you imagine our "BILL GATES" example in a 5 digit register, the decimal equivalent of our setup would look like this:

```
HIGH:  99999
LOW:   00000
```

In order to find our new range numbers, we need to apply the encoding algorithm from the previous section. We first had to calculate the range between the low value and the high value. The difference between the two registers will be 100000, not 99999. This is because we assume the high register has an infinite number of 9's added on to it, so we need to increment the calculated difference. We then compute the new high value using the formula from the previous section:

```
high = low + high_range(symbol)
```

In this case the high range was .30, which gives a new value for high of 30000. Before storing the new value of high, we need to decrement it, once again because of the implied digits appended to the integer value. So the new value of high is 29999.

The calculation of low follows the same path, with a resulting new value of 20000. So now high and low

look like this:

```
HIGH:   29999  (999...)
LOW:    20000  (000...)
```

At this point, the most significant digits of high and low match. Due to the nature of our algorithm, high and low can continue to grow closer to one another without quite ever matching. This means that once they match in the most significant digit, that digit will never change. So we can now output that digit as the first digit of our encoded number. This is done by shifting both high and low left by one digit, and shifting in a 9 in the least significant digit of high. The equivalent operations are performed in binary in the C implementation of this algorithm.

As this process continues, high and low are continually growing closer together, then shifting digits out into the coded word. The process for our "BILL GATES" message looks like this:

	HIGH	LOW	RANGE	CUMULATIVE OUTPUT
Initial state	99999	00000	100000	
Encode B (0.2-0.3)	29999	20000		
Shift out 2	99999	00000	100000	.2
Encode I (0.5-0.6)	59999	50000		.2
Shift out 5	99999	00000	100000	.25
Encode L (0.6-0.8)	79999	60000	20000	.25
Encode L (0.6-0.8)	75999	72000		.25
Shift out 7	59999	20000	40000	.257
Encode SPACE (0.0-0.1)	23999	20000		.257
Shift out 2	39999	00000	40000	.2572
Encode G (0.4-0.5)	19999	16000		.2572
Shift out 1	99999	60000	40000	.25721
Encode A (0.1-0.2)	67999	64000		.25721
Shift out 6	79999	40000	40000	.257216
Encode T (0.9-1.0)	79999	76000		.257216
Shift out 7	99999	60000	40000	.2572167
Encode E (0.3-0.4)	75999	72000		.2572167
Shift out 7	59999	20000	40000	.25721677
Encode S (0.8-0.9)	55999	52000		.25721677
Shift out 5	59999	20000		.257216775
Shift out 2				.2572167752
Shift out 0				.25721677520

Note that after all the letters have been accounted for, two extra digits need to be shifted out of either the high or low value to finish up the output word.

A Complication

This scheme works well for incrementally encoding a message. There is enough accuracy retained during the double precision integer calculations to ensure that the message is accurately encoded. However, there is potential for a loss of precision under certain circumstances.

In the event that the encoded word has a string of 0s or 9s in it, the high and low values will slowly converge on a value, but may not see their most significant digits match immediately. For example, high

and low may look like this:

```
High:     700004
Low:      699995
```

At this point, the calculated range is going to be only a single digit long, which means the output word will not have enough precision to be accurately encoded. Even worse, after a few more iterations, high and low could look like this:

```
High:     70000
Low:      69999
```

At this point, the values are permanently stuck. The range between high and low has become so small that any calculation will always return the same values. But, since the most significant digits of both words are not equal, the algorithm can't output the digit and shift. It seems like an impasse.

The way to defeat this underflow problem is to prevent things from ever getting this bad. The original algorithm said something like "If the most significant digit of high and low match, shift it out". If the two digits don't match, but are now on adjacent numbers, a second test needs to be applied. If High and low are one apart, we then test to see if the 2nd most significant digit in high is a 0, and the 2nd digit in low is a 9. If so, it means we are on the road to underflow, and need to take action.

When underflow rears its ugly head, we head it off with a slightly different shift operation. Instead of shifting the most significant digit out of the word, we just delete the 2nd digits from high and low, and shift the rest of the digits left to fill up the space. The most significant digit stays in place. We then have to set an underflow counter to remember that we threw away a digit, and we aren't quite sure whether it was going to end up as a 0 or a 9. The operation looks like this:

```
             Before    After
             ------    -----
High:        40344     43449
Low:         39810     38100
Underflow:   0         1
```

After every recalculation operation, if the most significant digits don't match up, we can check for underflow digits again. If they are present, we shift them out and increment the counter.

When the most significant digits do finally converge to a single value, we first output that value. Then, we output all of the "underflow" digits that were previously discarded. The underflow digits will be all 9s or 0s, depending on whether High and Low converged to the higher or lower value. In the C implementation of this algorithm, the underflow counter keeps track of how many ones or zeros to put out.

Decoding

In the "ideal" decoding process, we had the entire input number to work with. So the algorithm has us do

things like "divide the encoded number by the symbol probability." In practice, we can't perform an operation like that on a number that could be billions of bytes long. Just like the encoding process, the decoder can operate using 16 and 32 bit integers for calculations.

Instead of maintaining two numbers, high and low, the decoder has to maintain three integers. The first two, high and low, correspond exactly to the high and low values maintained by the encoder. The third number, code, contains the current bits being read in from the input bits stream. The code value will always lie in between the high and low values. As they come closer and closer to it, new shift operations will take place, and high and low will move back away from code.

The high and low values in the decoder correspond exactly to the high and low that the encoder was using. They will be updated after every symbol just as they were in the encoder, and should have exactly the same values. By performing the same comparison test on the upper digit of high and low, the decoder knows when it is time to shift a new digit into the incoming code. The same underflow tests are performed as well, in lockstep with the encoder.

In the ideal algorithm, it was possible to determine what the current encoded symbols was just by finding the symbol whose probabilities enclosed the present value of the code. In the integer math algorithm, things are somewhat more complicated. In this case, the probability scale is determined by the difference between high and low. So instead of the range being between 0.0 and 1.0, the range will be between two positive 16 bit integer counts. The current probability is determined by where the present code value falls along that range. If you were to divide (value-low) by (high-low+1), you would get the actual probability for the present symbol.

Where's the Beef?

So far, this encoding process may look interesting, but it is not immediately obvious why it is an improvement over Huffman coding. The answer becomes clear when we examine a case where the probabilities are a little different. Lets examine the case where we have to encode the stream "AAAAAAA", and the probability of "A" is known to be 0.90. This means there is a 90% chance that any incoming character will be the letter "A". We set up our probability table so that the letter "A" occupies the range 0.0 to 0.9, and the End of message symbol occupies the 0.9 to 1.0 range. The encoding process then looks like this:

New Character	Low value	High Value
	0.0	1.0
A	0.0	0.9
A	0.0	0.81
A	0.0	0.729
A	0.0	0.6561
A	0.0	0.59049
A	0.0	0.531441
A	0.0	0.4782969
END	0.43046721	0.4782969

Now that we know what the range of low and high values are, all that remains is to pick a number to encode this message. The number "0.45" will make this message uniquely decode to "AAAAAAA". Those two decimal digits take slightly less than 7 bits to specify, which means that we have encoded eight symbols in less than 8 bits! An optimal Huffman coded message would have taken a minimum of

9 bits using this scheme.

To take this point even further, I set up a test that had only two symbols defined. The byte value '0' had a 16382/16383 probability, and an EOF symbol had a 1/16383 probability of occurring. I then created a test file which was filled with 100,000 '0's. After compression using this model, the output file was only 3 bytes long! The minimum size using Huffman coding would have been 12,501 bytes. This is obviously a contrived example, but it does illustrate the point that arithmetic coding is able to compress data at rates much better than 1 bit per byte when the symbol probabilities are right.

The Code

An encoding procedure written in C is shown in listing 1 and 2. It contains two parts, an initialization procedure, and the encoder routine itself. The code for the encoder as well as the decoder were first published in an article entitled "Arithmetic Coding for Data Compression" in the February 1987 issue of "Communications of the ACM", by Ian H. Witten, Radford Neal, and John Cleary, and is being published here with the author's permission. I have modified the code slightly so as to further isolate the modeling and coding portions of the program.

There are two major differences between the algorithms shown earlier and the code in listing 1-2. The first difference is in the way probabilities are transmitted. In the algorithms shown above, the probabilities were kept as a pair of floating point numbers on the 0.0 to 1.0 range. Each symbol had its own section of that range. In the programs that are shown here, a symbol has a slightly different definition. Instead of two floating point numbers, the symbol's range is defined as two integers, which are counts along a scale. The scale is also included as part of the symbol definition. So, for the "BILL GATES" example, the letter L was defined previously as the high/low pair of 0.60 and 0.80. In the code being used here, "B" would be defined as the low and high counts of 6 and 8, with the symbol scale being 10. The reason it is done this way is because it fits in well with the way we keep statistics when modeling this type of data. The two methods are equivalent, it is just that keeping integer counts cuts down on the amount of work needing to be done. Note that the character actually owns the range up to but not including the high value.

The second difference in this algorithm is that all of the comparison and shifting operations are being done in base 2, rather than base 10. The illustrations given previously were done on base 10 numbers to make the algortihms a little more comprehensible. The algorithms work properly in base 10, but masking off digits and shifting in base 10 on most computers is difficult. So instead of comparing the top two digits, we now compare the top two bits.

There are two things missing that are needed in order to use the encoding and decoding algorithms. The first is a set of bit oriented input and output routines. These are shown in listing 3 and 4, and are presented here without comment. The modeling code is responsible for keeping track of the probabilities of each character, and performing two different transformations. During the encoding process, the modeling code has to take a character to be encoded and convert it to a probability range. The probability range is defined as a low count, a high count, and a total range. During the decoding process, the modeling code has to take a count derived from the input bit stream and convert it into a character for output.

Test Code

A short test program is shown Listing 3. It implements a compression/expansion program that uses the "BILL GATES" model discussed previously. Adding a few judiciously placed print statements will let

you track exactly what is happening in this program.

BILL.C has its own very simple modeling unit, which has a set of fixed probabilities for each of the possible letters in the message. I added a new character, the null string terminator, so as to know when to stop decoding the message.

BILL.C encodes an arbitrarily defined input string, and writes it out to a file called TEST.CMP. It then decodes the message and prints it to the screen for verification. There are a few functions in BILL.C that are modeling functions, which haven't been discussed yet. During the encoding process, a routine called convert_int_to_symbol is called. This routine gets a given input character, and converts it to a low count, high count, and scale using the current model. Since our model for BILL.C is a set of fixed probabilities, this just means looking up the probabilities in the table. Once those are defined, the encoder can be called.

During the decoding process, there are two functions associated with modeling. In order to determine what character is waiting to be decoded on the input stream, the model needs to be interrogated to determine what the present scale is. In our example, the scale (or range of counts) is fixed at 11, since there are always 11 counts in our model. This number is passed to the decoding routine, and it returns a count for the given symbol using that scale. Then, a modeling function called convert_symbol_to_int is called. It takes the given count, and determines what character matches the count. Finally, the decoder is called again to process that character out of the input stream.

Next Time

Once you successfully understand how to encode/decode symbols using arithmetic coding, you can then begin trying to create models that will give superior compression performance. Next month's concluding article discusses some of the methods you can try for adaptive compression. A fairly simple statistical modeling program is able to provide compression superior to respected programs such as PKZIP or COMPRESS.

References

As I mentioned previously, the article in the June 1987 "Communications of the ACM" provides the definitive overview of arithmetic coding. Most of the article is reprinted in the book "Text Compression", by Timothy C. Bell, John G. Cleary, and Ian H. Witten. This book provides an excellent overview of both statistical and dictionary based compression techniques. Another good book is "Data Compression" by James Storer.

Bell, Timothy C., Cleary, John G., Witten, Ian H, (1990) "Text Compression", Prentice Hall, Englewood NJ

Nelson, Mark (1989) "LZW Data Compression", Doctor Dobb's Journal, October, pp 29-37

Storer, J.A., (1988) "Data Compression", Computer Science Press, Rockville, MD

Witten, Ian H., Neal, Radford M., and Cleary, John G. (1987) "Arithmetic

Coding for Data Compression", Communications of the ACM, June, pp 520-540.

Listing 1 coder.h Constants, declarations, and prototypes needed to use the

		arithmetic coding routines. These declarations are for routines that need to interface with the arithmetic coding stuff in coder.c
Listing 2	coder.c	This file contains the code needed to accomplish arithmetic coding of a symbol. All the routines in this module need to know in order to accomplish coding is what the probabilities and scales of the symbol counts are. This information is generally passed in a SYMBOL structure.
Listing 3	bitio.h	This header file contains the function prototypes needed to use the bitstream i/o routines.
Listing 4	bitio.c	This routine contains a set of bit oriented i/o routines used for arithmetic data compression.
Listing 5	bill.c	This short demonstration program will use arithmetic data compression to encode and then decode a string that only uses the letters out of the phrase "BILL GATES". It uses a fixed, hardcoded table of probabilities.

Continue with Part 2: Statistical Modeling



Return to Mark Nelson's Home Page
Copyright (c) 1995-1998, Mark Nelson, All Rights Reserved.

This is the html version of the file http://www.cs.odu.edu/~mln/teaching/unc/inls210/?method=display&pkg_name=cryptolopes.pkg&element_name=cryptolopes.ppt.
Google automatically generates html versions of documents as we crawl the web.

IBM's Cryptolopes

http://www-4.ibm.com/software/security/cryptolope/

Overview

- A technology for securing intellectual property rights in a distributed environment
- Emphasizes security at the *document* level, not the channel level
- Developed for e-commerce settings, IBM has tuned cryptolopes for the digital library

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0971
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

market

History

- Developers

- Developed at IBM.
- Proposed by Jeff Crigler and Marc Kaplan [Benton 1996]
- Implemented by Kaplan, Josh Auerbach and Chee-Seng Chow
- Pursued by Ulrich Kohl, Jeffrey Lotspiech, and Marc A. Kaplan [Kohl et al. 1997]

History

- Time line

- 1994) Crigler proposes cryptolopes in the context of IBM's InfoMarket service (now defunct?). i.e. e-commerce origins:

The infoMarket service is using Cryptolopes to obtain both service revenue, through reselling a wide range of premium business information by the document, and licensing revenue, from companies

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0972
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

that set up Cryptolope clearinghouses for defined uses. [Benton 1996]

- 1995) Kaplan delivers initial prototype
- 1996?) IBM licenses Xerox PARC's Digital Property Rights Language (DPRL) for use in Cryptolopes
- 1997-98) "Safeguarding Digital Content" series appears in *Dlib* magazine, describing the motivation and architecture of cryptolopes.
- Current) Cryptolopes have been integrated into IBM's digital library management software.

Motivation

- Superdistribution

- Since distributing info is cheap, we should be able to broadcast it in bulk. i.e. Give information away, then charge for the rights to use it.
- e.g. Partially functional software downloads
- The idea of superdistribution is due to Ryoichi Mori. See Kaplan, Cox [1994].

Motivation

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0973
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- 4 goals of rights management for digital content
  - Authentication of both provider & reader
  - Authentication of content
  - Reader privacy
  - Superdistribution

[Kohl et al. (1997)]

Motivation

- Secure documents, open channels

- Traditional intellectual property protection invovles secure channels (SSL/SHHTP)
- This is fine for purposes of transmission
- But doesn't protect against illegal copying after legal transmission

Motivation

- Secure documents, open channels

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0974
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Instead of protecting the channel, Cryptolopes protect at the document level.
- Getting the document is free, but getting access to the document costs.
- So give the documents away! i.e. *superdistribution!*

Motivation

- Customizability

- Successful rights management depends on the context in which transactions take place.
- Content protection must be easily customizable.
- Rights management must be automated, and must scale well.

Motivation

Parties involved in content distribution

from http://www.dlib.org/dlib/may97/ibm/05gladney.html

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0975
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Analysis

- Basic idea of a cryptolope (cryptographic envelope):
  - Digital content is encrypted by the *creator*
  - Encrypted content is bundled into a cryptolope with non-encrypted (clear) metadata and associated authentication material
  - Encrypted content is freely distributed
  - Using clear metadata, *users* decide if they want to "license" the material
  - If so, users "purchase" decryption key from *publisher*

Architecture

- 3-tiers

- Kohl et al. (1997) and Gladney et al. (1997) describe rights management software utilizing 3 parties
- Creator
- Publisher

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0976
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Customer

Architecture

3-tiered distribution of digital content

from http://www.dlib.org/dlib/september97/ibm/09lotspiech.html

Architecture

- IBM (infoMarket) acts as a potential clearing house
- Creators may set up their own clearing house servers
- [Kohl, et al. (1997)] casts digital libraries as the clearing houses for publishers' cryptolopes

Architecture

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0977
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Structure of a cryptolope

from http://www.dlib.org/dlib/september97/ibm/09lotspiech.html

Architecture

- Cryptolope structure: important points
  - Cryptolope contains the keys to its own content
  - These keys are themselves encoded
  - Users buy keys for these keys from clearing house
  - i.e. clearing house only maintains keys for keys, not keys for content
  - Checksums assure authenticity of content

How does this work?

- If a user satisfies T/Cs, he is given the key

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0978
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Content is decrypted, shown in "special viewer"
- Content is authenticated using digital watermark
- Content is stamped with a user-specific digital fingerprint

Technical Details

- A cryptolope is a *.jar file
- Authentication via "X.509" certificates, a la Verisign…or you can create your own "lightweight certificates. Emphasis on providing just enough security for the distribution context.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0979
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

[See http://www-4.ibm.com/software/security/cryptolope/about.html for description]

System Requirements

- Cryptolope viewer application is available for Win32 (only successful under NT)
- The IBM Digital Library software runs under AIX, MacOS, and Win32. Presumably cryptolope server software does, too…but no mention in IBM docs
- IBM InfoMarket appears to be defunct (i.e. no readymade clearing house)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0980
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Summary

- Strengths

- Rights management that is flexible regarding:
  - Level of security
  - Terms and conditions
- Protecting at the document level allows liberal distribution

Summary

- Weaknesses

- Proprietary viewing software
- Reliance on long-term viability of clearing houses
- Lack of IBM support

References

Benton, P. (1996). "Packaging Information for Superdistribution" *IBM Research Magazine.* http://www.research.ibm.com/resources/magazine/1996/issue_3/packinginfo396.html

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0981
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Cox, Brad. (1994). "Superdistribution." *Wired.*
http://www.wired.com/wired/archive/2.09/superdis.html

Gladney, H. M. and Lotspiech, J. B. "Safeguarding Digital Library Contents and Users: Assuring Convenient Security and Data Quality." In *D-Lib Magazine,* May 1997.

http://www.dlib.org/dlib/may97/ibm/05gladney.html

Kaplan, Marc A. IBM Cryptolopes, Superdistribution and Digital Rights Management.
http://www.research.ibm.com/people/k/kaplan/cryptolope-docs/crypap.html

Ulrich Kohl, Jeffrey Lotspiech, and Marc A. Kaplan, "Safeguarding Digital Library Contents and Users: Protecting Documents Rather Than Channels", in *D-lib Magazine*, September, 1997.

http://www.dlib.org/dlib/september97/ibm/09lotspiech.html

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0982
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

本複製物は、特許庁が著作権法第４２条第２項第１号の規定により複製したものです。
取扱にあたっては、著作権侵害とならないよう十分にご注意ください国内技術雑誌2004-01180-006

Webの動画を速くきれいに！

WinPC Labs

コーデック

コーデックとファイルフォーマットを混同して理解していないだろうか。例えば、先日Microsoftが発表した「Windows Media Video 9 VCM」はWMVコーデックを使った映像データをAVIファイルに格納できるというものだ。今回は、こうした複雑な動画ファイルのフォーマットと、コーデックの関係を解説する。

文／竹松 昇
アイ・ビー・イーで各種MPEG応用システム、ストリーミングビデオシステムの開発に携わる。http://www.mpeg.co.jp/

AVIはストリーミングできない？ 動画ファイルの内部構造を知る

Microsoftのコーデック「Windows Media Video」を使うと、圧縮したビデオファイルがASFフォーマットのファイルに格納される。ところが最近、Microsoftから「Windows Media Video 9 VCM」というソフトウエアモジュールが発表された。これを使えばAVIフォーマットのファイルにWindows Media Videoで圧縮したビデオを格納できる。この内容をきちんと把握するには、コーデックとファイルフォーマットについて正しく理解する必要がある。

ファイルフォーマットとコーデックは、それぞれ独立して変更できるように設計されている。ファイルフォーマットは、ビデオやオーディオデータを格納する入れ物に相当する（図1）。ビデオとオーディオのデータを合わせて格納し、1つの「ファイル」としてコピーや再生、転送できるようにするものだ。ファイルフォーマットには、ビデオやオーディオの再生タイミングを規定して同期させるための情報や、ビデオの途中から再生できるようにするためのインデックス情報も格納されている。

コーデック（CODEC）は、COmpression/DECompressionの略で、圧縮と伸張をするソフトウエアやハードウエアのモジュールを意味する。ビデオとオーディオは別の手法、つまり別のコーデックで圧縮される。圧縮されたビデオデータとオーディオデータは、別々に存在する。これらの圧縮データを適切なファイルフォーマットに格納することで、1つのファイルとして取り扱うことができるようになる。

Windows Mediaを例にすると、図1のようになる。冒頭で説明したように、Windows Media＝ASFフォーマットとは限らない。以下では、よく使われるファイルフォーマットとコーデックの関係について解説しよう。

ストリーミングに向かないAVIフォーマット

AVIフォーマットは、Audio Video Interleaveの略で、Microsoftが1992年にApple ComputerのQuickTimeに対抗して開発

●ファイルとコーデックの関係

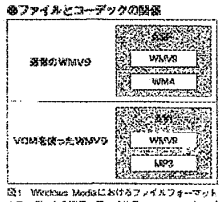


図1　Windows Mediaにおけるファイルフォーマットとコーデックの関係。ファイルフォーマットの中にビデオとオーディオのデータが格納される。

●AVIファイルの基本構造

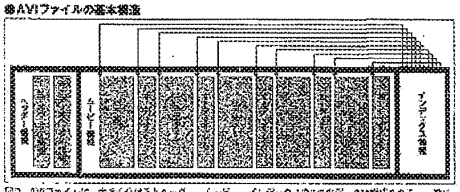


図2　AVIファイルは、大きく分けるとヘッダー、ムービー、インデックスの3つのデータで構成される。一般に、ビデオとオーディオのデータは交互に格納される。最後のインデックス情報から、再生すべきビデオ、オーディオデータの位置が分かる。いろいろなバリエーションがあり、作成ソフトによって異なる。

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0983
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

本複製物は、特許庁が著作権法第４２条第２項第１号の規定により複製したものです。
取扱にあたっては、著作権侵害とならないよう十分にご注意ください。

したものだ。技術的に時代遅れではあるが、現在でも多くのWindowsアプリケーションで使われている。最近ではDVファイルやDivXなどでおなじみだ。

AVIフォーマットは、RIFF（Resource Interchange File Format）と呼ばれる汎用の可変サイズフォーマットをベースにしている。音声ファイルのWAVフォーマットもRIFFをベースにしており、AVIフォーマットとWAVフォーマットは兄弟関係にある。

AVIフォーマットの構造を概略に図示したのが図2だ。ヘッダー情報、ムービー情報、インデックス情報の3つの領域で構成される。実際のビデオ、オーディオデータは、ムービー情報領域に交互に格納される。ヘッダー情報には、圧縮方式のほか、フレームレートなど再生に必要な情報を格納する。インデックス情報は、途中からの再生のため頭出しに必要な情報を持つ。

AVIフォーマットの再生には、一般にインデックス情報が必要となる。これはファイルの最後にあるため、ネットワークを経由してのストリーミング再生には向かない。また、複数の2つのフォーマットと異なり、ファイル中に時刻情報が存在しないため、長時間再生時のビデオ・オーディオの同期など、いくつかの点で不自由がある。

ASFフォーマットはインデックスなしでも再生

ASFフォーマットは1998年、MicrosoftがAVIフォーマットに変わる新しいファイルフォーマットとして規定した。Windows Mediaの標準ファイルフォーマットとなっている。当初は「Advanced Streaming Format」の略だったが、現在は「Advanced Systems Format」と、より汎用的な呼称となっている。

ASFフォーマットは、当初のお前通り、ストリーミング再生を意識して開発された。AVIフォーマット同様、インデックスオブジェクトを持つが、これがなくても再生できる。ビデオ・オーディオのデータそれぞれに再生時刻（先頭を0とした相対時刻）が付加しており、ビデオ・オーディオの同期が取れるためだ。ただし、インデックスオブジェクトがないと、途中からの再生のための頭出しはできない。

Windows Mediaのストリーミング配信では、先頭から順にヘッダーオブジェクト、ビデオ・オーディオデータが送信される。インデックスオブジェクトは送信されない。プレイヤーのスライダーで再生位置をジャンプした場合は、配信サーバーにコマンドが送られる。コマンドに従い、配信サーバー内でインデックスオブジェクトを読み、適切な位置から再度配信が始まる。

至る所に時間情報を持つMPEG-2 PSフォーマット

MPEGファイルフォーマットには、たくさん種類があるため、ここでは「MPEG-2 PS」について説明する。PSは「Program Stream」の略で、1995年に策定された。CDやDVDなどのメディアに格納することを意識している。実際にDVD-Videoをはじめ、多くのMPEG-2エンコーダーで使われている、もっとも普及したMPEG-2フォーマットだ。

MPEG-2 PSの場合、ビデオ・オーディオの圧縮データが交互に格納される点は、AVIフォーマットやASFフォーマットと同じ。違うのは、インデックス情報を持たないことだ。ファイルの途中から再生したい場合、ファイル中の至る所に格納されている時刻情報を手がかりにする。この方法でストリーミング再生も可能だ。

固定ビットレートの場合、およその位置が計算で求められるため、任意の位置を頭出しできる。一方、可変ビットレートの場合は、位置を計算で求めることができないため、辛いは難しい。頭出しのために、DVD-Videoではインデックス情報に相当する領域をSI全用意している。

また、PSでは「パック」と「PESパケット」と呼ぶ階層構造を持つ。パックは、CDやDVDのセクターを想定したもの。先頭にはファイルが読み込まれるべき時刻の情報があり、頭出しの手がかりとなる。PESパケットは、パックとは独立したビデオもしくはオーディオの格納単位だ。PESパケットの先頭には、再生されるべき時刻の情報が入っており、ビデオ・オーディオの同期をとる手がかりとする。

MPEG-2 PSには、通常、MPEG-2 VideoとMPEG-1 Audioで圧縮したデータを格納する。ほかのデータでも格納でき、DVD-Videoでは、オーディオにDolby Digitalやリニア PCMを使っている。

●ASFファイルの基本構造

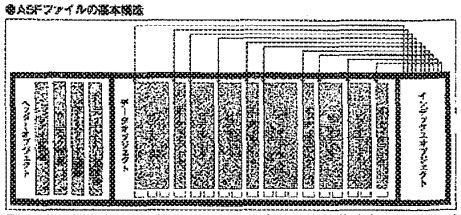


図3　ASFファイルは、大きく分けるとヘッダー、データ、インデックスの3つの領域で構成される。一般に、ビデオとオーディオのデータは、同一サイズのパケットに交互に格納される。各種のインデックス情報から、再生すべきビデオ、オーディオデータの位置が分かる。インデックス情報がなくても再生できるが、ストリーム途中へのジャンプができない。インデックス情報の間隔は自由だが、通常は数秒程度。

●MPEG-2 PSファイルの基本構造



図4　MPEG-2 PSファイルは、すべてのデータがパックとPESパケットという小さな単位に格納される。先頭のシステムヘッダーを除くと、残りはビデオ、オーディオのデータとなる。インデックス情報は持たないが、PESパケットの時刻情報を元に任意の場所にジャンプできる。図ではパック、PESパケット構造は簡略化して表記した。パックとPESパケットの境界は一致しなくてもよく、それぞれの長さも可変でよい。

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0984
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

XP-002617671

Matroska File Format (under construction!)

Alexander Noé
alex@alexander-noe.com
Last change: June 24, 2007

Contents

1 Introduction 4

2 EBML - basics 6
- 2.1 Unsigned Integer Values of Variable Length ("vint") 6
- 2.2 EBML elements 7
- 2.3 Signed Integer Values of Variable Length (svint) 7
- 2.4 Data Types 8

3 MATROSKA files - Top-Level elements 9
- 3.1 EBML 9
- 3.2 Segment 9

4 EBML - The EBML file header 10

5 Level 1 - Elements inside Segments 12
- 5.1 Overview 12
- 5.2 SegmentInfo 13
- 5.3 SeekHead 16
- 5.4 Tracks 18
- 5.5 Cluster 28
- 5.6 Cues 30
- 5.7 Chapters - Editions and ChapterAtoms 32
- 5.8 Attachments 37
- 5.9 Tags 38

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0985
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

6 MATROSKA block Layout and Lacing 42
6.1 Basic layout of a Block . . . 42
6.2 Lacing . . . 43

7 Overhead of MATROSKA files 44
7.1 Overhead of BLOCKGROUPS . . . 44
7.2 Overhead of CLUSTERS . . . 49
7.3 Overhead caused by Cues . . . 50

8 Links 51

List of Tables

1 EBML . . . 10
2 Segment . . . 12
3 SegmentInfo . . . 14
4 SeekHead . . . 16
5 Seek . . . 16
6 Tracks . . . 18
7 TrackEntry . . . 18
8 Video . . . 21
9 Audio . . . 22
10 ContentEncodings . . . 23
11 ContentEncoding . . . 23
12 ContentCompression . . . 24
16 Cluster . . . 28
17 BlockGroup . . . 29
18 Cues . . . 31
19 CuePoint . . . 31
20 CueTrackPositions . . . 31
21 Chapters . . . 32
22 EditionEntry . . . 32
23 ChapterAtom . . . 34

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0986
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

24 ChapterTracks 35
25 ChapterDisplay 36
26 Attachments 37
27 AttachedFile 37
28 Tags 38
29 Tag 39
30 Targets 40
31 SimpleTag 41

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0987
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1 Introduction

This document is intended to be used by developers who want to implement support for the MATROSKA file format in their applications, but who want to build this support from scratch rather than using existing implementations, or people who just want to understand the MATROSKA file format in detail. Thus, the file format itself is described, the usage of existing libraries isn't.

This document does not replace the official documentation[1]. It is less condensed, but not necessarily complete. Especially, in the case that MATROSKA supports Digital Restrictions Management one day, I will expressively not document that part. Also, typos in element IDs are never impossible.

When speaking about element occurence, elements can be mandatory or not, elements may be present several times inside a parent element or not etc. Occurence restrictions will be indicated using expressions like $= 1$ or ≥ 1 etc. Those restrictions will exclude cases which do not technically render a file unusable or ambigous, but which are unreasonable, like a file with no SEGMENTUID, see section 5.2. The same way it would be weird (but not make a file unusable) to have a CHAPTERS element (which is supposed to describes chapters) which is empty. An element that must occur at least once is a reasonable file is called "mandatory". When an element is really mandatory, i.e the file or a part of it is useless when it's missing, it will be labeled as ≥ 1 (!) or $= 1$ (!). An example would be the codec ID of a track, without which a track cannot be decoded at all.

The official Matroska specification pages use the following interpretation of "mandatory" and "default": When an element has a default value that is used if the element itself is not present, the value cannot be missing, thus the element is inherently mandatory. This interpretation of "mandatory" being weird, this document considers an element mandatory when it must be physically present in the file. Also, default values can only be valid values. Consequently, a mandatory element cannot have a default value because if it had one, it couldn't be mandatory anymore.

In this document, element names are always printed like THIS, element values are printed like $THIS, as in "if $THISFLAG=1, ...".

If you have any questions concerning this document, if you have comments, additions, if you have found an error, or if you want to contact me for whatever reason,

[1] http://www.matroska.org/technical/specs/index.html

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0988
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

please send me an e-mail (include 'matroska' in the topic!). You can contact me in german, english or french, whatever you prefer. Just don't ask me if you can ask something or if I could document some Digital Restrictions Management.

This document is powered by LaTeX, so changing the order of certain tables or the style of those tables etc. is, with certain limits, possible within a few seconds.

Screenshots of real life file structures are used to illustrate the file structure. All of them have been made using the EBML Tree Viewer in AVI-Mux GUI.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0989
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2 EBML - basics

EBML files use integers of variable size. This way, the file format doesn't waste space with storing 32 or even 64 bit integers in placed where they *might sometimes* occur. The way the size is coded is inspired by the UTF-8 encoding format.

2.1 Unsigned Integer Values of Variable Length ("vint")

The length of an integer is equivalent to $length = 1 + |number\_of\_leading\_zero\_bits|$. All integers use big endian. You could use more than 7 leading zeros, then the first byte would be 0x00, however, this would only be needed if integers longer than 56 bits are required. This is forbidden in MATROSKA files.

Example: 3A 41 FE:

The first byte 3A (0011 1010) has 2 leading zeros, resulting in a total length of 3 bytes. The first '1' in the byte (0011 1010) is just needed to finish the sequence of leading zeros and can't be used to store the value either. Thus, it is reset to obtain the value this byte sequence represents. The result is then 0x1A41FE. As you can see, you lose one bit per byte to know how long a number is, and you can use 7 bits per byte to store the integer's value itself.

Of course, the value 0x1A41FE could also be written as 10 1A 41 FE or 08 00 1A 41 FE (do the decoding on a piece of paper if it's not clear), however, when writing EBML files, the shortest possible encoding should be used to avoid wasting space, which is the very point of this coding scheme.

Unknown Length

All bits after the leading zeros being set to one, such as FF or 7F FF, indicates an *unknown length*. Muxers shall avoid writing unknown length values whenever possible. The only exception is the last Level 0 element of a file. If encoding a number as described above results in such a sequence, it must be encoded again with a greater destination length. Example: When encoding 16383 as described above, the result is 7F FF. In 7F FF, all bits after the leading zero are set, which would indicate an unknown length. That means, the length is increased to 3, and the number is encoded again to 20 3F FF.

Note

It is possible to use a lookup table to determine the total length from the first byte.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0990
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The Matroska file format does not allow integer lengths greater than 8, meaning that the number of leading zeros is not higher than 7 and that the total length can always be retrieved from the first byte.

2.2 EBML elements

One piece of information is stored the following way:

```
typedef struct {
    vint      ID        // EBML-ID
    vint      size      // size of element
    char[size] data     // data
} EBML_ELEMENT;
```

The length of ID shall be called `s_ID`, the length of size shall be called `s_size`. Elements that contain other EBML Elements are called *EBML Master elements*.

Generally, the order of EBML elements inside a parent element is not fixed. In some cases, a certain order is recommended, but it is never mandatory. Especially, no element order should be assumed inside small parent elements.

2.3 Signed Integer Values of Variable Length (svint)

Signed integers have the following value: Read the integer as Unsigned Integer and then subtract

```
vsint_subtr[length-1]
```

where

```
__int64 vsint_subtr [] =
    { 0x3F, 0x1FFF, 0x0FFFFF, 0x07FFFFFF,
      0x03FFFFFFFF, 0x01FFFFFFFFFF,
      0x00FFFFFFFFFFFFFF, 0x007FFFFFFFFFFFFF };
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0991
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2.4 Data Types

Whereas `vints` are used in the header section of EBML elements, the data types describes in this section occur in the data section.

2.4.1 Signed and Unsigned Integers (int and uint)

Integers, signed as well as unsigned, are stored in big endian byte order, with leading 0x00 (in case of positive values) and 0xFF (in case of negative values) being cut off (example for int: -257 is 0xFE 0xFF). An int/uint may not be larger than 8 bytes.

2.4.2 Float

A `Float` value is a 32 or 64 bit real number, as defined in IEEE. 80 Bit values have been in the specification, but have been removed and should not be used. The bytes are stored in big endian order.

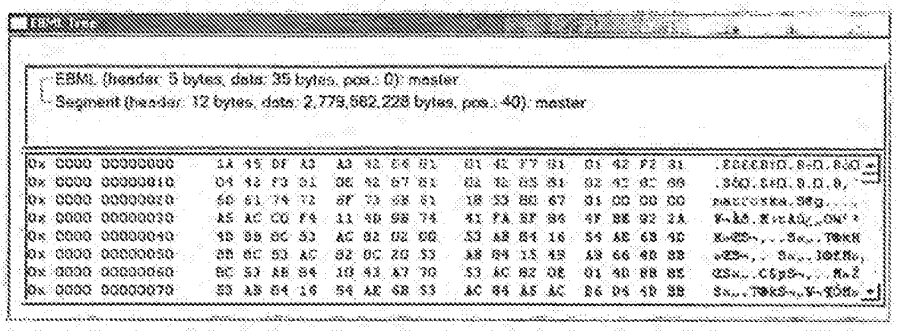
2.4.3 Types of Strings

String refers to an ASCII string.
UTF-8 refers to a string that is encoded as UTF-8

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0992
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3 MATROSKA files - Top-Level elements

MATROSKA files only have two different top level elements:



3.1 EBML

This header describes the contents of an EBML file. There should be only one EBML header in one file. Any further EBML headers do not render a file invalid, but shall be ignored by any application reading the file. Files with more than one EBML header could be created for instance if two or more files are appended by using the copy /b command.

3.2 Segment

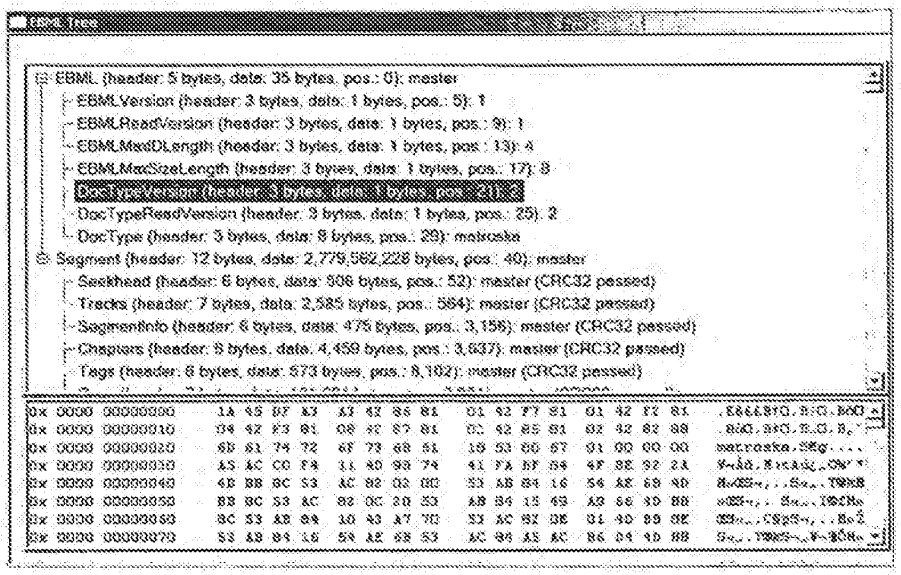
A SEGMENT contains multimedia data, as well as any header data necessary for replay. There can be several SEGMENTs in one MATROSKA file, but this is not encouraged to be done, as not many tools are able to handle multisegment MATROSKA files correctly. If you want to replay multisegment MATROSKA files on Windows, please use Haali Media splitter[2]

[2] http://haali.cs.msu.ru/mkv/

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0993
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

4 EBML - The EBML file header

The EBML top level element contains a description of the file type, such as EBML version, file type name, file type version etc.



Obviously, this header being missing makes it necessary to guess the file type.

Table 1: The EBML element (Top-Level)	
Element	Description
uint, # ≤ 1 EBMLVERSION ID: 42 86 def: 1	indicates the version of the EBML Writer that has been used to create a file
uint, # ≤ 1 EBMLREADVERSION ID: 42 F7 def: 1	indicates the minimum version an EBML parser needs to be compliant with to be able to read the file

EBML *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0994
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
uint, # ≤ 1 EBMLMAXIDLENGTH ID: 42 F2 def: 4	indicates the length of the longest EBML-ID the file contains. In case of matroska, this value is 4. Any EBML-ID which is longer than the value of this element shall be considered invalid.
uint, # ≤ 1 EBMLMAXSIZELENGTH ID: 42 F3 def: 8	indicates the maximum `s_size` value the file contains. Any EBML element having an `s_size` value greater than EBML-MaxSizeLength shouldl be considered invalid.
string, # ≤ 1 DOCTYPE ID: 42 82 def: matroska	describes the contents of the file. In the case of a MATROSKA file, its value is `'matroska'`
uint, # ≤ 1 DOCTYPEVERSION ID: 42 87 def: 1	indicates the version of the $DOCTYPE writer used to create the file
uint, # ≤ 1 DOCTYPEREADVERSION ID: 42 85 def: 1	indicates the minimum version number a $DOCTYPE parser must be compliant with to read the file.

Index →page 2 *end of* EBML

As you can see, in the case of Matroska files all child elements of the EBML element have a default value. Thus, an empty EBML element would technically introduce a Matroska file (with file type version 1, maximum ID length 4, maximum size length 8 etc.) correctly. However, I don't recommend to push the specifications like this.

It is not recommended to use either IDs or size values greater than 8 bytes. While it's clear that 8 bytes are enough to represent any size of anything on any hard disc, one might think about using IDs larger than 8 bytes. However, since the ID is considered an integer, treating IDs larger than 8 bytes is difficult on current CPUs, which are limited to 64 bit for simple integer operations.

BNSDOCID: <XP _____3617671A_1_>

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0995
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5 Level 1 - Elements inside Segments

5.1 Overview

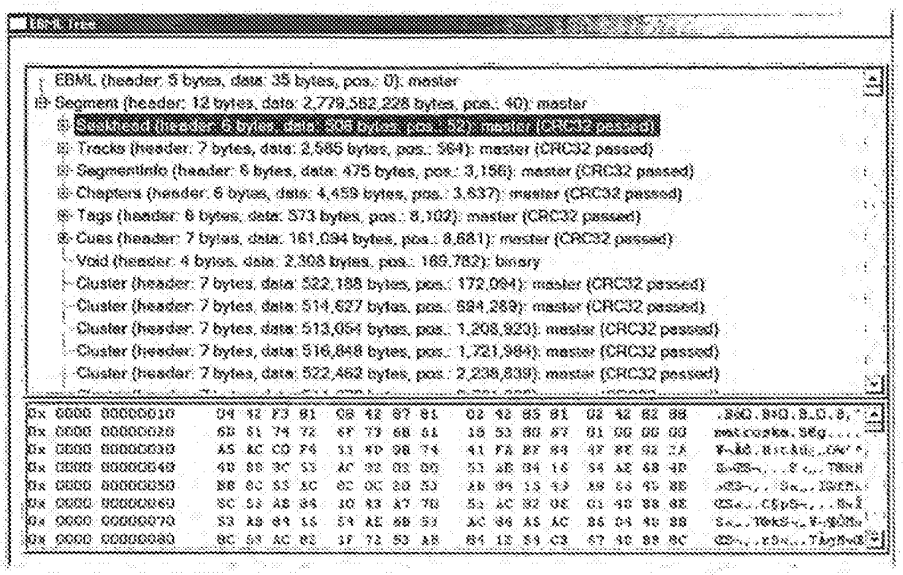


Table 2: The SEGMENT element (Top-Level)	
Element	Description
Master, # = 1 SEGMENTINFO (→3) ID: 15 49 A9 66	SEGMENTINFO contains general information about a segment, like an UID, a title etc. This information is not really required for playback, but should be there (→ section 5.2).
Master, # ≥ 0 SEEKHEAD (→4) ID: 11 4D 9B 74	A SEEKHEAD is an index of elements that are children of SEGMENT. It can point to other SEEKHEADS, but not to itself. If all non-CLUSTER precede all CLUSTERS (→ section 5.5), a SEEKHEAD is not really necessary, otherwise, a missing SEEKHEAD leads to long file loading times or the inability to access certain data.

SEGMENT *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0996
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

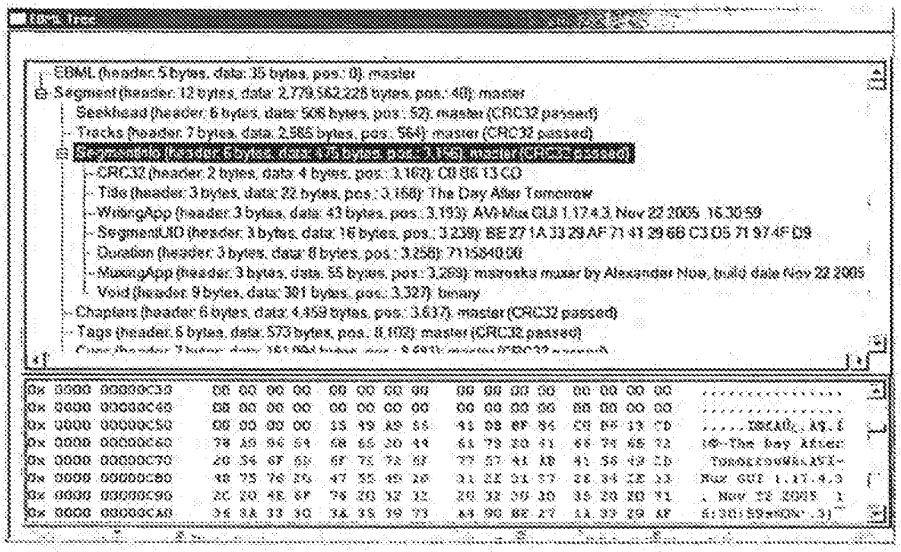
Element	Description
Master, # ≥ 0 CLUSTER (→16) ID: 1F 43 B6 75	A CLUSTER contains video, audio and subtitle data. Note that a MATROSKA file could contain chapter data or attachments, but no multimedia data, so CLUSTER is not a mandatory element.
Master, # ≥ 0 TRACKS (→6) ID: 16 54 AE 6B	A TRACKS element contains the description of some or all tracks (preferably all). This element can be repeated once in a while for backup purposes. A file containing only chapters and attachments does not have a TRACKS element, thus it's not mandatory.
Master, # ≤ 1 CUES (→18) ID: 1C 53 BB 6B	The CUES element contains a timestamp-wise index to CLUSTERS, thus it's helpful for easy and quick seeking.
Master, # ≤ 1 ATTACHMENTS (→26) ID: 19 41 A4 69	The ATTACHMENTS element contains all files attached to this SEGMENT.
Master, # = 1 CHAPTERS (→21) ID: 10 43 A7 70	The CHAPTERS elements contains the definition of all chapters and editions of this SEGMENT
Master, # ≤ 1 TAGS (→28) ID: 12 54 C3 67	The TAGS element contains further information about the SEGMENT or elements inside the SEGMENT that is not really required for playback.

Index →page 2 *end of* SEGMENT

5.2 SegmentInfo

The SEGMENTINFO element contains general information about the SEGMENT, such as its duration, the application used for writing the file, date of creation, a unique 128 bit ID, to name a few only. Information included in the SEGMENTINFO element is not required for playback, but should be written by any MATROSKA muxer.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0997
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



(mkvi: <element name> (<s_size> + <_ID>) : <size> bytes at <position in file>: value)

Table 3: The SEGMENTINFO element, child of SEGMENT (→2)	
Element	Description
char[16], # = 1 SEGMENTUID ID: 73 A4	a unique 128 bit number identifying a SEGMENT. Obviously, a file can only be referred to by another file if a SEGMENTUID is present, however, playback is possible without that UID.
utf-8, # ≤ 1 SEGMENTFILENAME ID: 73 84	contains the name of the file the SEGMENT is stored in. Since renaming files is easy, the reliability of this element's value should not be overrated.
char[16], # ≤ 1 PREVUID ID: 3C B9 23	contains the unique 128 bit ID of the SEGMENT that is replayed before the currently active SEGMENT, i.e. the ID of the SEGMENT that should be loaded if the user tries to seek to a timecode earlier than the earliest timecode of the active SEGMENT. That SEGMENT should, of course, be easy to locate, for instance in a file in the same directory.

SEGMENTINFO *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0998
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
utf-8, # ≤ 1 PREVFILENAME ID: 3C 83 AB	contains the name of the file in which the SEGMENT having the ID $PREVUID is stored. PREVFILENAME should not be considered reliable for the same reason as SEGMENTFILENAME, however, it could be the first filename the player is looking for when the SEGMENT described in PREVUID is needed
char[16], # ≤ 1 NEXTUID ID: 3E B9 23	contains the unique 128 bit ID of the SEGMENT that is replayed after the currently active SEGMENT, i.e. the ID of the SEGMENT that should be loaded if the user tries to seek to a timecode after the end of the active SEGMENT. Like PREVUID, the corresponding SEGMENT should be easy to locate.
utf-8, # ≤ 1 NEXTFILENAME ID: 3E 83 BB	contains the name of the file in which the SEGMENT having the ID $NEXTUID is stored. NEXTFILENAME shall not be considered reliable for the same reason as SEGMENTFILENAME.
uint, # ≤ 1 TIMECODESCALE ID: 2A D7 B1	Each scaled timecode in a MATROSKA file is multiplied by TIMECODESCALE to obtain a timecode in nanoseconds. Note that not all timecodes are scaled!
float, # ≤ 1 DURATION ID: 44 89	The DURATION indicates the duration of the SEGMENT. The duration measured in nanoseconds is scaled and is thus equal to $DURATION * $TIMECODESCALE. This element should be written.
utf-8, # ≤ 1 TITLE ID: 7B A9	Contains a general name of the SEGMENT, like "Lord of the Rings - The Two Towers". No language can be attached to the title, however, Tags (→ section 5.9) could be used to define several titles for a segment. This is not yet commonly done, though.
string, # = 1 MUXINGAPP ID: 4D 80	contains the name of the library that has been used to create the file (like "libmatroska 0.7.0"). This element should be written by any muxer! Especially if non-compliant files are encountered, this help to know who must be blamed for that file.
utf-8, # = 1 WRITINGAPP ID: 57 41	contains the name of the application used to create the file (like "mkvmerge 0.8.1"). This element should be written for the same reason as MUXINGAPP.

SEGMENTINFO *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 0999
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
int, # ≤ 1 DATEUTC ID: 44 61	contains the production date, measured in nanoseconds relatively to Jan 01, 2001, 0:00:00 GMT+0h

Index →page 2 *end of* SEGMENTINFO

5.3 SeekHead

The SEEKHEAD element contains a list of positions of Level 1 elements in the SEGMENT. Each pair (element id, position) is stored in one SEEK element:

Table 4: The SEEKHEAD element, child of SEGMENT (→2)	
Element	Description
Master, # ≥ 1 SEEK (→5) ID: 4D BB	One SEEK element contains an EBML-ID and the position within the SEGMENT at which an element with this ID can be found.

Index →page 2 *end of* SEEKHEAD

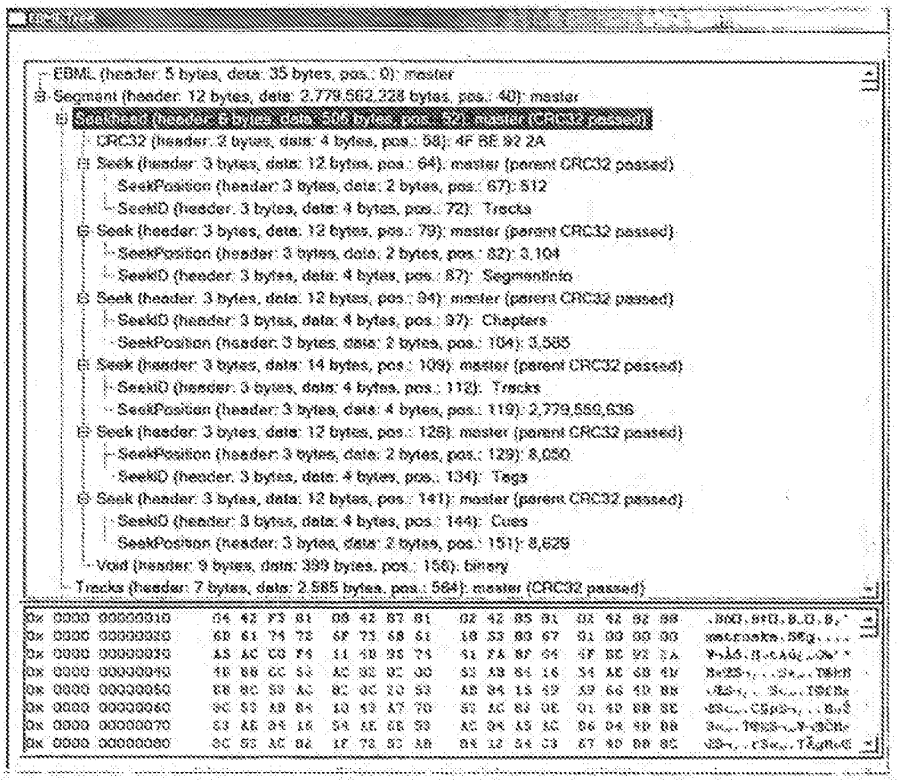
Table 5: The SEEK element, child of SEEKHEAD (→4)	
Element	Description
uint, # = 1 SEEKID ID: 53 AB	The SEEKID element contains the EBML-ID of the element found at the given position
uint, # = 1 SEEKPOSITION ID: 53 AC	The SEEKPOSITION element contains the position relatively to the SEGMENT's data at which an element with the ID $SEEKID can be found.

Index →page 2 *end of* SEEK

Not all Level 1 elements need to be included. Typical SEEKHEADs either include a list of all Level 1 elements, or a list of all Level 1 elements except for CLUSTERS (→ section 5.5). SEEKHEADs can also include references to other SEEKHEADs if there is, for example, a small SEEKHEAD at the beginning of the file and a larger one at its end.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1000
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The following picture illustrates the SEEKHEAD element in a real file. Note that the EBML Tree Viewer replaced Level 1 IDs in SEEKID with their human-readable name:



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1001
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5.4 Tracks

The TRACKS element contains information about the tracks that are stored in the SEGMENT, like track type (audio, video, subtitles), the used codec, resolution and sample rate. All tracks shall be described in one (or more, but preferably only one) TRACKS element.

Each track is described in one TRACKENTRY. Theoretically, using the TRACKUID, information about one track could be spread over different TRACKENTRYs, the UID would allow to know which track the information applies to, however, it is highly discouraged to stretch the specification like this.

Also, an empty TRACKS element would be rather useless, but should not lead to a parser error since the file can be played if all tracks are defined *somewhere*. Especially pure chapter files might have an empty TRACKS element if the muxer doesn't catch the case that no tracks are present and consequently creates an empty TRACKS element.

An example of a TRACKENTRY element can be found on (→ page 26)

Table 6: The TRACKS element, child of SEGMENT (→2)	
Element	Description
Master, # ≥ 1 TRACKENTRY (→7) ID: AE	One TRACKENTRY element describes one track of the SEGMENT

Index →page 2 *end of* TRACKS

Table 7: The TRACKENTRY element, child of TRACKS (→6)	
Element	Description
uint, # = 1 (!) TRACKNUMBER ID: D7	defines an identification number of the track. This number cannot be equal to 0. This number is used by the BLOCK and SIMPLEBLOCK structures.
uint, # = 1 TRACKUID ID: 73 C5	is a unique identificator of the track within the file. It cannot be equal to 0

TRACKENTRY *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1002
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
uint, # = 1 (!) TRACKTYPE (→13) ID: 83	defines the type of a track, i.e. video, audio, subtitle etc.
bool, # ≤ 1 FLAGENABLED ID: B9 def: 1	When FLAGENABLED is 1, track is used
bool, # ≤ 1 FLAGDEFAULT ID: 88 def: 1	When FLAGDEFAULT is 1, the track should be selected by the player by default. Obviously, if no video track and/or no audio track has a default flag, one video track and one audio track should be chosen by the player, whereas no subtitle should be enabled if no subtitle has a default flag.
bool, # ≤ 1 FLAGFORCED ID: 55 AA def: 0	When FLAGFORCED is 1, the track must be played. When several subtitle tracks are forced, the one matching the audio language should be chose. An example would be a subtitle track that cannot be disabled, like the one you find on the german DVD "Eiskalte Engel" when you select english audio. Since this flag can only be used to apply a restriction on digital content, it must be qualified as Digital Restrictions Management.
bool, # ≤ 1 FLAGLACING ID: 9C def: 0	When FLAGLACING is 1, the track may contain laced blocks. A parser that supports all types of lacing (→ section 6.2) can safely ignore this flag.
uint, # ≤ 1 MINCACHE ID: 6D E7 def: 0	indicates the number of frames a player must be able to cache during playback. This is for instance interesting if a native MPEG4 file with frames in coding order is played.
uint, # ≤ 1 MAXCACHE ID: 6D F8	indicates the maximum cache size a player needs to cache frames. A value of NULL means that no cache is required.

TRACKENTRY *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1003
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
uint, # ≤ 1 DEFAULTDURATION ID: 23 E3 83	This value indicates the number of nanoseconds a frame lasts. This value is applied if no $DURATION value is indicated for a frame or if lacing (→ section 6.1) is used. A value of 0 means that the duration of frames of the track is not necessarily constant (e.g. variable framerate video, or Vorbis audio). DEFAULTDURATION should be written for each track with a constant frame rate since it makes seeking easier.
float, # ≤ 1 TRACKTIMECODESCALE ID: 23 31 4F	Every timecode of a block (`cluster timecode + block timecode`) is multiplied by this value to obtain the real timecode of a block.
utf-8, # ≤ 1 NAME ID: 53 6E	A NAME element contains a human-readable name for the track. Note that you can't define which language this track name is in. You have to use Tags (→ section 5.9)) if you want to use several titles in different languages for the same track.
string, # ≤ 1 LANGUAGE ID: 22 B5 9C def: eng	specifies the language of a track, using ISO639-2[3]. This is NOT necessarily the language of $NAME, for example a german AC3 track could be called "German - AC3 5.1" or "Deutsch - AC3 5.1" or "Allemand AC3 5.1" etc.
string, # = 1 (!) CODECID ID: 86	The CODECID specifies the Codec[4] which is used to decode the track.
binary, # ≤ 1 CODECPRIVATE ID: 63 A2	CODECPRIVATE contains information the codec needs before decoding can be started. An example is the Vorbis initialization packets for Vorbis audio.
utf-8, # ≤ 1 CODECNAME ID: 25 86 88	CODECNAME is a human-readable name of the Codec
uint, # ≥ 0 ATTACHMENTLINK ID: 74 46	An ATTACHMENTLINK contains the UID of an attachment that is used by this track.

TRACKENTRY *continued on next page*

[3] http://loveb.loc.gov/standards/iso639-2/englangn.html
[4] http://matroska.org/technical/specs/codecid/index.html

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1004
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
Master, # ≤ 1 VIDEO (→8) ID: E0	VIDEO contains information that is specific for video tracks
Master, # ≤ 1 AUDIO (→9) ID: E1	AUDIO contains information that is specific for audio tracks
Master, # ≤ 1 CONTENTENCODINGS (→10) ID: 6D 80	CONTENTENCODINGS contains information about (lossless) compression or encryption of the track

Index →page 2 *end of* TRACKENTRY

Obviously, the VIDEO element must be present for video tracks, whereas the AUDIO element must be present for audio tracks. Although it doesn't make sense to have both elements in one TRACKENTRY element, it wouldn't make a file unplayable.

Table 8: The VIDEO element, child of TRACKENTRY (→7)

Element	Description
uint, # = 1 PIXELWIDTH ID: B0	Width of the encoded video track in pixels
uint, # ≤ 1 PIXELHEIGHT ID: BA	Height of the encoded video in pixels
uint, # ≤ 1 PIXELCROPBOTTOM ID: 54 AA def: 0	Number of Pixels to be cropped from the bottom
uint, # ≤ 1 PIXELCROPTOP ID: 54 BB def: 0	Number of Pixels to be cropped from the top

VIDEO *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1005
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
uint, # ≤ 1 PIXELCROPLEFT ID: 54 CC def: 0	Number of Pixels to be cropped from the left
uint, # ≤ 1 PIXELCROPRIGHT ID: 54 DD def: 0	Number of Pixels to be cropped from the right
uint, # ≤ 1 DISPLAYWIDTH ID: 54 B0 def: $PIXELWIDTH	Width of the video during playback
uint, # ≤ 1 DISPLAYHEIGHT ID: 54 BA def: $PIXELHEIGHT	Height of the video during playback
uint, # ≤ 1 DISPLAYUNIT ID: 54 B2 def: 0	Unit $DISPLAYWIDTH and $DISPLAYHEIGHT is measured in. This can be 0→pixels, 1→centimeters, 2→inches

Index →page 2 *end of* VIDEO

$PIXELCROPXXXX is applied on $PIXELXXX, so the output is cropped after decoding, but before stretching it to the dimensions indicated with $DISPLAYXXXX.

Table 9: The AUDIO element, child of TRACKENTRY (→7)

Element	Description
uint, # ≤ 1 SAMPLINGFREQUENCY ID: B5 def: 8 kHz	Indicates the sample rate the track is encoded at in Hz

AUDIO *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1006
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
uint, # ≤ 1 OUTPUTSAMPLINGFREQUENCY ID: 78 B5	Indicates the sample rate the track must be played at in Hz. The default value of this element is equal to $SAMPLINGFREQUENCY.
uint, # ≤ 1 CHANNELS ID: 9F def: 1	Number of channels of the audio track
uint, # ≤ 1 BITDEPTH ID: 62 64	Bits per sample, this is usually used with PCM-Audio.

Index →page 2 *end of* AUDIO

Table 10: The CONTENTENCODINGS element, child of TRACKENTRY (→7)

Element	Description
Master, # ≥ 1 CONTENTENCODING (→11) ID: 62 40	A CONTENTENCODING-element describes one compression or encryption that has been used on this track.

Index →page 2 *end of* CONTENTENCODINGS

Table 11: The CONTENTENCODING element, child of CONTENTENCODINGS (→10)

Element	Description
uint, # ≤ 1 CONTENTENCODINGORDER ID: 50 31 def: 0	Tells when to decode according to this pattern. The decoder starts with the CONTENTENCODING that has the highest CONTENTENCODINGORDER.

CONTENTENCODING *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1007
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
uint, # ≤ 1 CONTENTENCODING-SCOPE (→14) ID: 50 32 def: 1	Defines which parts of the track are compressed or encrypted this way
uint, # ≤ 1 CONTENTENCODING-TYPE ID: 50 33 def: 0	Describes which type of encoding is described. 0 → compression, 1 → encryption
Master, # ≤ 1 CONTENTCOMPRESSION (→12) ID: 50 34	If CONTENTENCODINGTYPE=0, this element describes how it is compressed
Master, # ≤ 1 CONTENTENCRYPTION (→??) ID: 50 35	If CONTENTENCRYPTION=1, this element describes how it is encrypted

Index →page 2 *end of* CONTENTENCODING

The CONTENTENCODING element allows to apply not only encryption, but also lossless compression to a track. This can be used to compress text subtitles, but also to remove sync headers from audio packers. For example, each AC3 frame starts with 0B 77, and there is no real point in saving those two bytes for each frame in a MATROSKA file. For a simple AC3 file, this does make sense because there it can be used to find a new frame start if data is damaged.

Table 12: The CONTENTCOMPRESSION element, child of CONTENTENCODING (→11)

Element	Description
uint, # ≤ 1 CONTENTCOMPALGO (→15) ID: 42 54 def: 0	The CONTENTCOMPALGO element says which algorithm was used for this compression.

CONTENTCOMPRESSION *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1008
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
binary, # ≤ 1 CONTENTCOMPSETTINGS ID: 42 55	Contains settings that are required for decompression. These settings are specific for each compression algorithm. For example, it contains the striped header bytes when $CONTENTCOMPALGO=3 (→ page 26).

Index →page 2 *end of* CONTENTCOMPRESSION

Table 13: Values of TRACKTYPE, child of TRACKENTRY (→7)

Value	Description
0x01	track is a video track
0x02	track is an audio track
0x03	track is a complex track, i.e. a combined video and audio track
0x10	track is a logo track
0x11	track is a subtitle track
0x12	track is a button track
0x20	track is a control track

end of TRACKTYPE

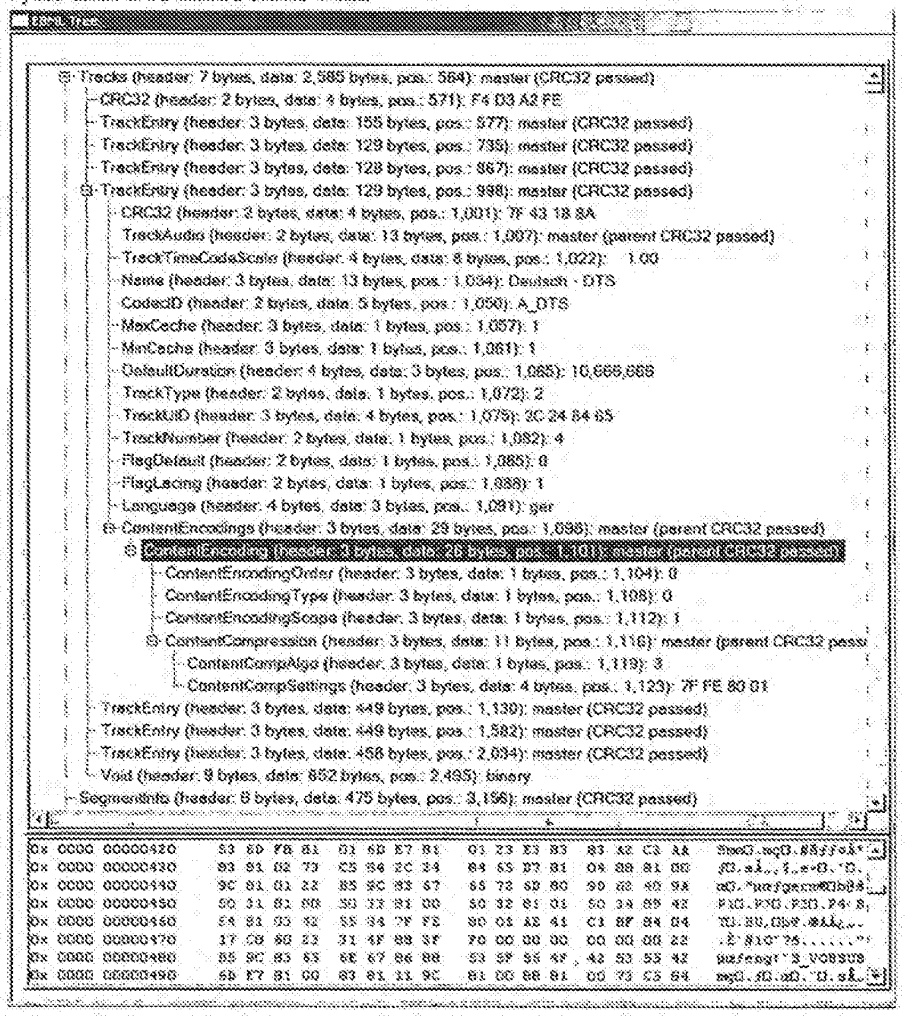
Table 14: Bits in CONTENTENCODINGSCOPE, child of CONTENTENCODING (→11)

Value	Description
1	all frames
2	the track's CODECPRIVATE
4	the CONTENTCOMPRESSION in the next CONTENTENCODING (next as in next in decoding order)

end of CONTENTENCODINGSCOPE

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1009
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Here is one example of a possible TRACKENTRY element: A DTS-audio track that is using header striping. The CONTENTCOMPSETTINGS element contains the four bytes each DTS frame starts with.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1010
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Table 15: Values of CONTENTCOMPALGO, child of CONTENTCOMPRESSION (→12)

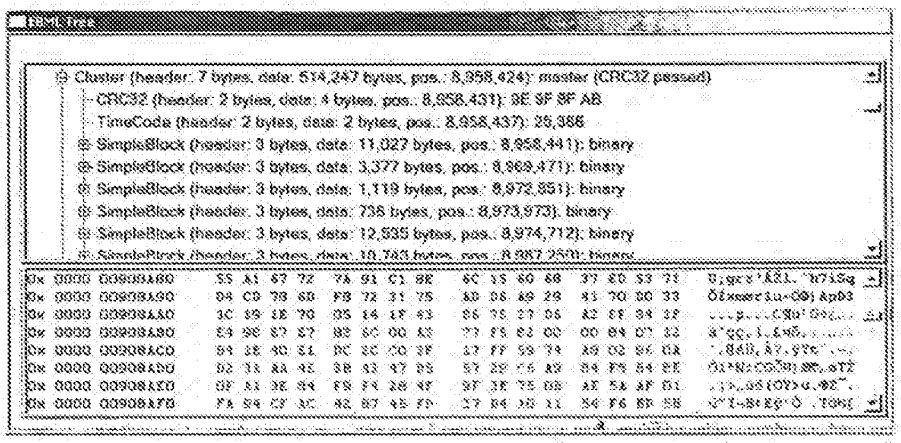
Value	Description
0	zlib
1	bzlib
2	lzo1x
3	header striping

end of CONTENTCOMPALGO

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1011
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5.5 Cluster

A CLUSTER contains multimedia data and usually spans over a range of a few seconds. The following picture shows a typical cluster:



Although sticking to this order of the elements is not mandatory, it is recommended not to have any non-BLOCKGROUP/SIMPLEBLOCK after the first BLOCKGROUP/SIMPLEBLOCK, because it's bad if the entire cluster must be read before it can be used just because the timecode is stored at the end.

Table 16: The CLUSTER element, child of SEGMENT (→2)	
Element	Description
uint, # ≤ 1 TIMECODE ID: E7 def: 0	The Cluster timecode is the timecode all block timecodes are indicated relatively to.
uint, # ≤ 1 POSITION ID: A7	The POSITION element indicates the position of the beginning of its parent element inside its grand parent element. This can help to resync in case of damaged data, but is of no use if no data is damaged.

CLUSTER *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1012
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
uint, # ≤ 1 PREVSIZE ID: AB	Indicates the size of the preceding cluster in bytes. This helps to seek backwards, and to find the preceding cluster, without having to look at METASEEK or CUE data. This is also helpful to resync, e.g. if the EBML-ID of the preceding CLUSTER is damaged.
Master, # ≥ 0 BLOCKGROUP (→17) ID: A0	Contains a BLOCK along with some attached information like references
binary, # ≥ 0 SIMPLEBLOCK ID: A3	This is a BLOCK (→ page 42) without additional attached information. Since a SIMPLEBLOCK does not require a BLOCKGROUP around it, it causes less overhead. SIMPLEBLOCK is MATROSKA v2.

Index →page 2 *end of* CLUSTER

Table 17: The BLOCKGROUP element, child of CLUSTER (→16)

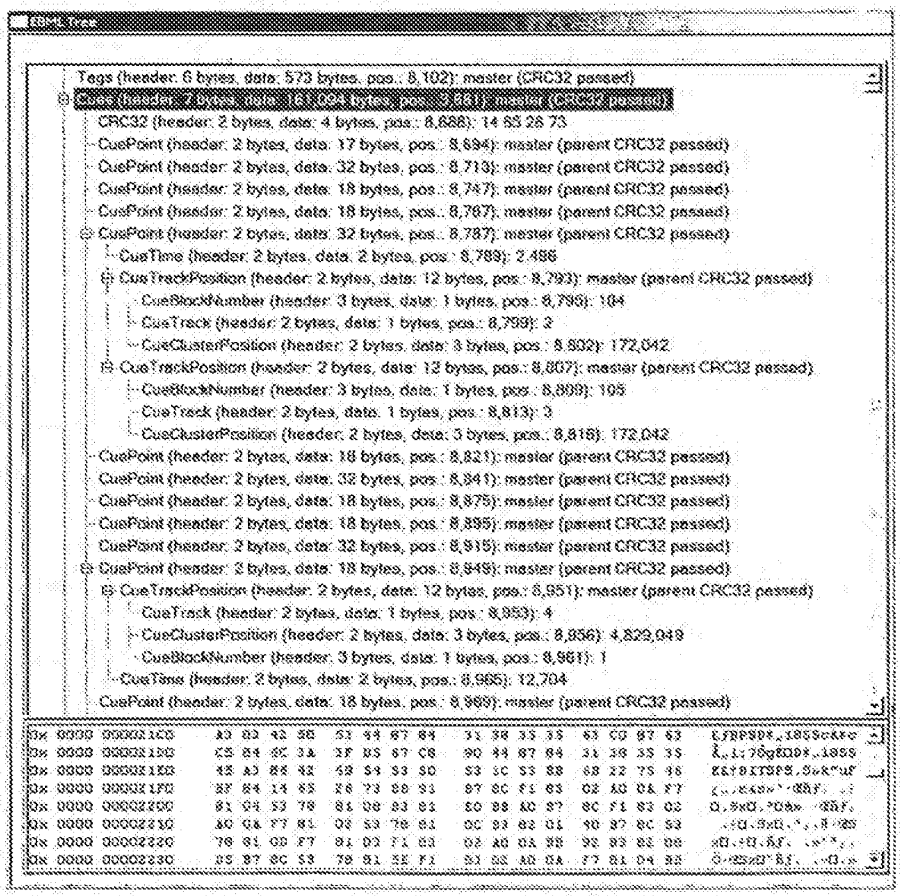
Element	Description
binary, # = 1 (!) BLOCK ID: A1	contains data to be replayed. See page 42 for details.
int, # ≥ 0 REFERENCEBLOCK ID: FB	Timecode of a frame, relative to the BLOCK's timecode, of a frame that needs to be decoded before this BLOCK can be decoded.
int, # ≤ 1 BLOCKDURATION ID: 9B	Indicates the scaled duration of the BLOCK. If this value is not written, it is assumed to be (1) the difference `<timecode of next block of the same stream> - <timecode>` (2) equal to DEFAULTDURATION (for the last block of each stream). As a consequence, the DURATION element is mandatory for every BLOCK of subtitle tracks, unless a subtitle is indeed supposed to disappear only directly before the next one appears. But even then it is recommended to write DURATION.

Index →page 2 *end of* BLOCKGROUP

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1013
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5.6 Cues

The CUES element contains information helpful (but not necessary) for seeking. Each piece of information, called a CUEPOINT, contains a timestamp, and a list of pairs (track number, (cluster position[, block number within cluster])). Generally, a CUEPOINT should only point to keyframes.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1014
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Table 18: The CUES element, child of SEGMENT (→2)	
Element	Description
Master, # ≥ 1 CUEPOINT (→19) ID: 0xBB	One CUEPOINT contains one entry point (or a list of entry points with one point for one track) for one timecode.

Index →page 2 *end of* CUES

Table 19: The CUEPOINT element, child of CUES (→18)	
Element	Description
uint, # = 1 (!) CUETIME ID: 0xB3	The timecode of the CLUSTERS or BLOCKS that are referred to by this CUEPOINT
Master, # ≥ 1 CUETRACKPOSITIONS (→20) ID: 0xB7	A position where a CLUSTER or BLOCK can be found with the timecode $CUETIME.

Index →page 2 *end of* CUEPOINT

Table 20: The CUETRACKPOSITIONS element, child of CUEPOINT (→19)	
Element	Description
uint, # ≥ 1 (!) CUETRACK ID: 0xF7	Track for which a position is given. This track number is the same as TRACKENTRY (→ Table 7)::TRACKNUMBER.
uint, # ≥ 1 (!) CUECLUSTERPOSITION ID: 0xF1	The position of the cluster the referred block is found in. This position is relative to the SEGMENT's (→ Table 2) data section.
uint, # ≤ 1 CUEBLOCKNUMBER ID: 0x53 0x78	The block with timecode $CUETIME is the $CUEBLOCKNUMBER-th BLOCK/SIMPLEBLOCK inside the CLUSTER at position $CUECLUSTERPOSITION.

Index →page 2 *end of* CUETRACKPOSITIONS

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1015
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5.7 Chapters - Editions and ChapterAtoms

The CHAPTERS element contains a list of all editions and chapters found in this SEGMENT. Chapters in MATROSKA files are more powerful than chapters on DVDs, their handling is, however, way more complex.

Table 21: The CHAPTERS element, child of SEGMENT (→2)	
Element	Description
Master, # ≥ 1 EDITIONENTRY (→22) ID: 45 B9	One EDITIONENTRY describes one Edition. Just like with TRACKENTRY (→ Table 7), theoretically you could spread information about one Edition over different EDITIONENTRYS and use $EDITIONUID to find out which edition the EDITIONENTRY is referring to, but it's highly discouraged.

Index →page 2 *end of* CHAPTERS

An edition contains one set of chapter definitions, so having several editions means having several sets of chapter definitions. This case is used when using this as a playlist - playing one chapter after the other while having gaps between the chapters.

Table 22: The EDITIONENTRY element, child of CHAPTERS (→21)	
Element	Description
uint, # ≤ 1 EDITIONUID ID: 45 BC	$EDITIONUID is the UID of the edition. This element is mandatory if you want to apply one or more titles to an edition
bool, # ≤ 1 EDITIONFLAGHIDDEN ID: 45 BD def: 0	When $EDITIONFLAGHIDDEN is 1, this edition should not be available via the user interface
bool, # ≤ 1 EDITIONFLAGDEFAULT ID: 45 DB def: 0	When $EDITIONFLAGDEFAULT is 1, this edition should be selected by the player as default

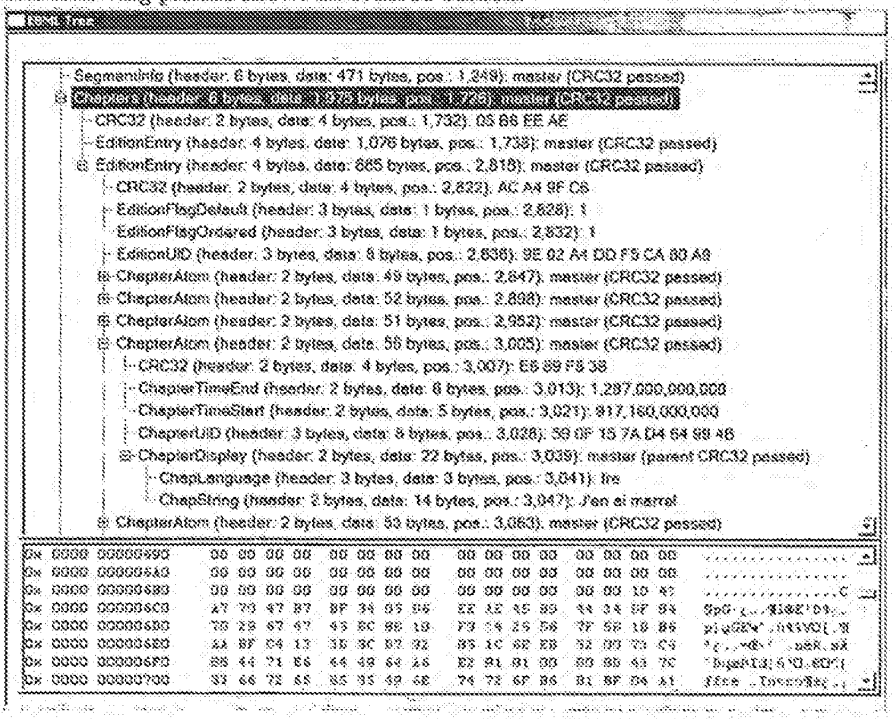
EDITIONENTRY *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1016
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
bool, # $\leq$ 1 EDITIONFLAGORDERED ID: 45 DD def: 0	When $EDITIONFLAGORDERED is 1, this edition contains a playlist. When $EDITIONFLAGORDERED is 0, it contains a simple DVD like chapter definition.
Master, # $\geq$ 1 CHAPTERATOM (→23) ID: B6	One CHAPTERATOM contains the definition of one chapter. This element is the only one in MATROSKA files that can contain itself recursively - in this case to define subchapters.

Index →page 2 *end of* EDITIONENTRY

The following picture shows an ordered edition:



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1017
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Table 23: The CHAPTERATOM element, child of EDITIONENTRY (→22), child of CHAPTERATOM (→23)

Element	Description
uint, # = 1 CHAPTERUID ID: 73 C4	The UID of this chapter. It must be unique within the file.
uint, # ≤ 1 CHAPTERTIMESTART ID: 91 def: 0	The *unscaled* timecode the chapter starts at. As the value is unsigned, a chapter cannot start earlier than at timecode 0, even whereas timecodes up to -30.000 are possible for multimedia data.
uint, # ≤ 1 CHAPTERTIMEEND ID: 92	The *unscaled* timecode the chapter ends at. The default value is the start of the next chapter or the end of the parent chapter or the end of the segment, whatever exists, in that order.
bool, # ≤ 1 CHAPTERFLAGHIDDEN ID: 98 def: 0	When $CHAPTERFLAGHIDDEN is 1, the chapter should not be visible in the user interface, but should be played back normally.
bool, # ≤ 1 CHAPTERFLAGENABLED ID: 45 98 def: 1	When $CHAPTERFLAGENABLED is 0, the chapter should be skipped by the player
char[16], # ≤ 1 CHAPTERSEGMENTUID ID: 6E 67	This element can only occur if $EDITIONFLAGORDERED=1. The SEGMENT of which the UID is $CHAPTERSEGMENTUID is used instead of the current SEGMENT. Obviously, this SEGMENT should be easy to find, like when it is the first segment of a file in the same directory.
uint, # ≤ 1 CHAPTERSEGMENT -EDITIONUID ID: 6E BC	The edition to use inside the SEGMENT selected via CHAPTERSEGMENTUID. The timecodes $CHAPTERTIMESTART and $CHAPTERTIMEEND refer to playback timecodes of that edition, i.e. the timecodes are relative to that playlist. This is called "nested Editions" and is NOT SUPPORTED by *Haali Media Splitter*.

CHAPTERATOM *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1018
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
Master, # ≤ 1 CHAPTERTRACKS (→24) ID: 8F	Contains a list of tracks the chapter applies to.
Master, # ≥ 0 CHAPTERDISPLAY (→25) ID: 80	Contains all chapter titles

Index →page 2 *end of* CHAPTERATOM

A useful application for the CHAPTERFLAGHIDDEN element in connection with ordered editions is the following: You have a couple of episodes of a series, but want to save space by only saving the intro and outtro once. You create one playlist (ordered edition) per episode, and another playlist playing all episodes in a row. Whereas in the first case you might want to play intro and outtro for each episode, you might not want to do that in the second case.

If you don't want to make the three parts intro - movie - outtro selectable via the user interface when playing single episodes, you call the intro-chapter "Episode - blah" and hide the movie- and the outtro chapter using $CHAPTERFLAGHIDDEN=1. Then, the playlist playing all episodes would be *intro - episode 1 - episode 2 - ... - last episode - outtro*, whereas the other playlists would be *intro - episode N - outtro*. The name of the intro chapter would be set to "Episode *n*".

Table 24: The CHAPTERTRACKS element, child of CHAPTERATOM (→23)

Element	Description
uint, # ≥ 1 CHAPTERTRACKNUMBER ID: 89	One number of a track a chapter is used with.

Index →page 2 *end of* CHAPTERTRACKS

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1019
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Table 25: The CHAPTERDISPLAY element, child of CHAPTERATOM (→23)	
Element	Description
utf-8, # ≤ 1 CHAPSTRING ID: 85	A title of a chapter
string, # ≥ 0 CHAPLANGUAGE ID: 43 7C def: eng	The language of $CHAPSTRING as defined in ISO639-2[5]
utf-8, # ≥ 0 CHAPCOUNTRY ID: 43 7E	A country the title is used in. For example, a german title in Germany might be different than the title used in Austria.

Index →page 2 *end of* CHAPTERDISPLAY

[5] http://lcweb.loc.gov/standards/iso639-2/englangn.html#two

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1020
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5.8 Attachments

Theoretically, any file type can be attached to a MATROSKA file, however, this possibility is usually used to attach pictures like CD covers or fonts required to display a subtitle track correctly. Obviously, attaching executable files would allow for MATROSKA files to contain viruses - a scenario that is not exactly the indended application of attachments or anything else MATROSKA is capable of.

Table 26: The ATTACHMENTS element, child of SEGMENT (→2)	
Element	Description
Master, # ≥ 1 ATTACHEDFILE (→27) ID: 61 A7	Describes and contains one attached file

Index →page 2 *end of* ATTACHMENTS

Table 27: The ATTACHEDFILE element, child of ATTACHMENTS (→26)	
Element	Description
utf8, # ≤ 1 FILEDESCRIPTION ID: 46 7E	A human-readable description of the file
utf8, # ≤ 1 FILENAME ID: 46 6E	The name that should be proposed by a demuxer when extracting the file
string, # ≤ 1 FILEMIMETYPE ID: 46 60	MIME type of the file, like ...
binary, # ≤ 1 FILEDATA ID: 46 5C	The file itself
uint, # = 1 FILEUID ID: 46 AE	The UID of that file, just like TRACKUID, CHAPTERUID etc. The UID is required if a TRACKENTRY (→ Table 7) wants to refer to this Attachment.

Index →page 2 *end of* ATTACHEDFILE

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1021
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5.9 Tags

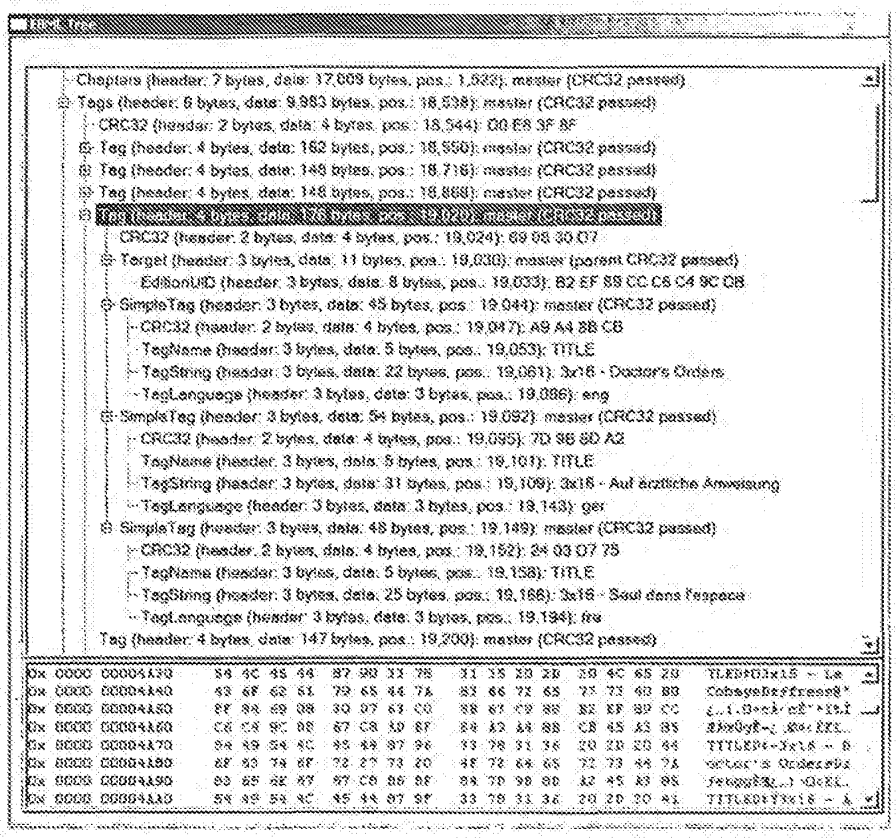
Table 28: The TAGS element, child of SEGMENT (→2)	
Element	Description
Master, # ≥ 1 TAG (→29) ID: 73 73	One TAG element describes one Tag

Index →page 2 *end of* TAGS

TAGS provide additional information[6] not important for replay. A TAGS element contains a number of TAG elements. Each TAG element contains a list of UIDs (usually TRACKUIDs or EDITIONUIDs), and a list of SIMPLETAGs, each one containing a name and a value:

[6] http://www.matroska.org/technical/specs/tagging/index.html

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1022
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



If no TARGETS are specified, then the TAG is a global TAG refering to the entire SEGMENT. Of course, two different TAG elements can contain identical TARGETS.

Table 29: The TAG element, child of TAGS (→28)	
Element	Description
Master, # ≤ 1 TARGETS (→30) ID: 63 C0	Describes which elements a Tag applies to

TAG *continued on next page*

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1023
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Element	Description
Master, # ≥ 1 SIMPLETAG (→31) ID: 67 C8	Each SIMPLETAG contains one tag that applies to each target in TARGETS

Index →page 2 *end of* TAG

Note that there is nothing like a TAGUID.

Table 30: The TARGETS element, child of TAG (→29)

Element	Description
uint, # ≤ 1 TARGETTYPEVALUE (→??) ID: 68 CA def: 50	This number describes the logical level of the object the Tag refers to
utf-8, # ≤ 1 TARGETTYPE ID: 63 CA	A string describing the logical level of the object the Tag is refering to
uint, # ≥ 0 TRACKUID ID: 63 C5	The UID of a track the tag is referring to
uint, # ≥ 0 EDITIONUID ID: 63 C9	The UID of an edition the tag is referring to. Note that this is the only way to apply titles to an edition
uint, # ≥ 0 CHAPTERUID ID: 63 C4	The UID of a chapter the tag is referring to
uint, # ≥ 0 ATTACHMENTUID ID: 63 C6	The UID of an attachment the tag is referring to

Index →page 2 *end of* TARGETS

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1024
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Table 31: The SIMPLETAG element, child of TAG (→29)	
Element	Description
utf-8, # ≥ 1 (!) TAGNAME ID: 45 A3	Name of the tag.
string, # ≤ 1 TAGLANGUAGE ID: 44 7A def: und	$TAGLANGUAGE is the language of $TAGNAME. Note that the default here is 'und', whereas the default track / chapter title language is 'eng'.
bool, # ≤ 1 TAGORIGINAL ID: 44 84 def: 1	When 1, this title and language is the original title given to the item
utf-8, # ≤ 1 TAGSTRING ID: 44 87	The value of the tag when it is a string
binary, # ≤ 1 TAGBINARY ID: 44 85	The 'value' of the tag when it's a binary tag

Index →page 2 *end of* SIMPLETAG

5.9.1 A few common Tags

- TITLE, Target: EditionUID: used to define names for Editions. This is exactly what you can see in the screenshot above.
- BPS, Target: TrackUID: used to define the bitrate of a track
- FPS, Target: TrackUID: used to define the framerate of a track

BNSDOCID: <XP_____2617671A_1_>

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1025
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

6 MATROSKA block Layout and Lacing

6.1 Basic layout of a Block

A MATROSKA block has the following format:

```
BLOCK {
  vint TrackNumber
  sint16 Timecode  // relative to Cluster timecode
  int8 Flags      // lacing, keyframe, discardable
  if (lacing) {
    int8 frame_count-1
    if (lacing ==  EBML lacing) {
       vint size[0]
       svint size[1..frame_count-2]
    } else
    if (lacing == Xiph lacing) {
      int8 size[size of <leading (frame_count-1) frames> / 255 + 1]
    }
  }
  int8[] data
}
```

The following bits are defined for FLAGS:

```
  Bit  0x80: keyframe:
             No frame after this frame can reference any frame before
             this frame and vice versa (in AVC-words: this frame is an
             IDR frame). The frame itself doesn't reference any other
             frames.
  Bits 0x06: lace type
             00 - no lacing
             01 - Xiph lacing
             11 - EBML lacing
             10 - fixed-size lacing
  Bit 0x08 : invisible: duration of this block is 0
  Bit 0x01 : discardable: this frame can be discarded if the decoder
             is slow
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1026
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The following flags are only defined for Matroska v2 and can thus only be used in a SIMPLEBLOCK: *keyframe, invisible, discardable*. The type of lacing in use defines how the SIZE values are to be read.

6.2 Lacing

Lacing is a technique that allows to store more than one atom of data (like one audio frame) in one block, with the goal to decrease overhead, without losing the ability to separate the frames in a lace later again.

Generally, the size of the last frame in a Lace is not stored, as it can be derived from the total block size, the size of the block header and the sum of the sizes of all other frames.
Frame duration values are not preserved! That means, it is highly recommended not to use lacing if the frame duration is not constant, like Vorbis audio.

6.2.1 Xiph Lacing

The size of each frame is coded as a sum of int8. A value smaller than 255 indicates that the next value refers to the next frame.

Example
`size = { 187, 255, 255, 120, 255, 0, 60 }` means that there are 4 frames with `187, 630, 255, 60` bytes.

6.2.2 EBML Lacing

Size of first frame ("frame 0") of a lace = `size[0]`
Size of frame `i` of a lace: `size[i] - size[i-1]`

6.2.3 Fixed Lacing

Fixed Lacing is used if all frames in a lace have the same size. Examples are AC3 or DTS audio. In this case, knowing the number of frames is enough to calculate the size of one frame. Consequently, there are no `size` values.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1027
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

7 Overhead of MATROSKA files

The scope of this section is explaining how to predict the overhead of a MATROSKA file before muxing, and without analysing any of the source files excessively. This section assumes that BLOCKGROUPS and BLOCKS are used, and that no SIMPLEBLOCKS are used. If you want to estimate overhead of files that use SIMPLEBLOCKS, you get about the same overhead as with BLOCKS without BLOCKDURATION, REFERENCEBLOCK or BLOCKGROUP.

7.1 Overhead of BLOCKGROUPS

First, here again the layout of a typical BLOCKGROUP

```
BlockGroup <size>
   Block <size> <number, flag, timecode>
   [ Reference <size> <val> ]
```

The EBML identication for BLOCKS and BLOCKGROUPS are 1 byte each, so that the structure above, not counting REFERENCES, takes:

- BlockGroup < 128 bytes: **8 bytes**
- BlockGroup < 16kbytes: **10 bytes**
- BlockGroup < 2MBytes: **12 bytes**

BLOCKGROUPS larger than 2MBytes are extremely unlike, and even BLOCKGROUPS larger than 16kBytes won't occur often, compared to BLOCKGROUPS between 128 bytes and 16 kBytes. That means, assuming an overhead of 10 bytes for BLOCKGROUPS without REFERENCES usually results in a good approximation.

7.1.1 video

In a typical video stream, there are a lot of frames with 1 REFERENCE (P-Frames, Delta-Frames), and a few keyframes. Typical rations are 100:1. There might also be frames with 2 REFERENCES (B-Frames), e.g. native MPEG4 streams. Assuming a ratio of 66:33:1 for B:P:K, and assuming a bitrate far below 3,2 MBit/s (meaning that typical B- and P-frames are smaller than 16 kB), that causes about 15 bytes of

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1028
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

overhead per frame. If there are no B-Frames, there are about 13 bytes per frame.

Example: 2 hours, 25 fps.
The video stream will cause around 2,3 MB of overhead.

7.1.2 audio - without lacing

As audio does usually not have any REFERENCES (all audio frames are keyframes), one audio frame will take 8 or 10 bytes of overhead. For MP3, AC3, DTS and AAC, frames causing 8 bytes of overhead are unlikely. They are more likely for Vorbis.

Example: MP3 audio, 24ms per frame, duration: 2h
This stream will cause 3MB of overhead.

7.1.3 audio - with lacing

1. CBR+CFR: fixed lacing
In this case, *fixed lacing* (see section 6.2.3) is used. With fixed lacing, the overhead is the normal BLOCKGROUP overhead, plus 1 byte for the lace header. Assuming that BLOCKGROUPS are not larger than 16k, that means that the overhead per frame is equal to `11 / frame_count`

Example: AC3 audio, 448 kbps, 1792 bytes per frame, 32ms per frame
1.) 8 frames per lace.
overhead for one frame = `11/8 = 1,375 bytes = 1 byte / 23,3 ms.`
2.) 9 frames per lace.
overhead for one frame = `11/9 = 1,222 bytes = 1 byte / 26,2 ms.`
3.) 10 frames per lace.
overhead for one frame = `13/10 = 1,3 bytes = 1 byte / 24,6 ms.`

An AC3 stream of 2 hours with 9 frames per lace will cause 270kB of overhead.

2. no CBR, but almost all frames smaller than 255 bytes: XIPH lacing
In this case, XIPH lacing (see section 6.2.1) is used, meaning that the overhead of a BLOCKGROUP is equal to `normal BlockGroup overhead + frame_count`, meaning that the overhead per frame is about `(11+frame_count)/frame_count`, if there are `frame_count` frames in each lace. Again, if the BLOCKGROUPS are larger than 16kBytes, then the overhead is `(13+frame_count)/frame_count`.
In other words, the ratio in `bytes / frame` will always be between about 1,2 and

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1029
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2,5 for audio streams with mainly small frames.

Although XIPH lacing is also defined for larger frames, EBML lacing is usually more effective then.

3. otherwise: EBML lacing Assuming that the difference in size between 2 consecutive frames is smaller than 8191, 1 or 2 bytes are needed to code the size of each frame, additionally to the normal BLOCKGROUP overhead.

As a result, we get 3 possible estimations:

a) worst case That means, a lace with `frame_count` frames using EBML lacing will cause not more than `((11 or 13)+2*frame_count)/frame_count` bytes of overhead per frame.

Example 1: 16 frames per lace, BLOCKGROUP > 16kB, worst case:
`overhead <= (13 + 2*16)/16 = 2,8 bytes / frame.`
Example 2: 8 frames per lace, BLOCKGROUP < 16kB, worst case:
`overhead <= (11 + 2*8)/8 = 3,4 bytes / frame.`

b) best case The best case is obviously that 2 consecutive frames differ by not more than 62 bytes. In that case, one byte is needed to code the size of one frame. However, the first frame might need to bytes, if it is larger than 126 bytes.

Example 1: 16 frames per lace, BLOCKGROUP > 16kB, best case:
`overhead <= (13 + 1*16)/16 = 1,8 bytes / frame.`
Example 2: 8 frames per lace, BLOCKGROUP < 16kB, best case:
`overhead <= (11 + 1*8)/8 = 2,4 bytes / frame.`

c) average case This is the case you need for optimal overhead prediction. Unfortunately, the average case depends on the compression format of the corresponding audio track, its bitrate, maybe even the encoder that has been used. The easiest way to gather data on the average case of EBML lace header overhead is to simulate the lace results of different files that are likely to be used. Candidates are MPEG 1/2/4 audio and Vorbis, but not AC3 or DTS.
I have run a simulation with the following file types:
MPEG 1 Layer 3 (128 and 192 kbps, 48 kHz), HE-AAC (224 kbps and 96 kbps, 44,1 kHz), LC-AAC (268 kbps, 44,1 kHz)

The results obtained from those files are discussed on the following pages. The lace behaviour simulation has been run using `mls`[7] (short for 'matroska lace sim-

[7] `http://www-user.tu-chemnitz.de/~noe/Video-Zeug/mls/`

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1030
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

ulator"). Note that it would be required to run the simulation and to evaluate the results as follows for each audio format, in each bitrate, maybe even with each encoder, for which results as accurate as possible shall be predicted.

The results for the lace header size are as follows:

Audio Format	Lace header overhead per frame @ <x> Frames per lace								
	4	8	12	16	24	32	48	64	96
MP3 @ 128 kbps	1,39	1,29	1,26	1,24	1,22	1,22	1,21	1,20	1,20
MP3 @ 192 kbps	1,50	1,41	1,38	1,37	1,36	1,35	1,34	1,34	1,33
HE-AAC @ 224 kbps	1,39	1,29	1,25	1,24	1,22	1,21	1,20	1,20	1,20
HE-AAC @ 64 kbps	1,34	1,23	1,19	1,18	1,16	1,15	1,14	1,14	1,13
LC-AAC @ 268 kbps	1,31	1,19	1,16	1,14	1,12	1,11	1,10	1,09	1,09

Applications using `libmatroska` for MATROSKA file creation are using 8 frames per lace. As a consequence, the overhead for a track using EBML lacing can be predicted to an acceptable accuracy if the audio format is known.
As you can also see, larger laces hardly affect the overhead caused by the lace headers of BLOCKs from a certain size on.

However, larger laces mean fewer BLOCKs and thus fewer BLOCKGROUPs, so the total overhead per frame, including the overhead caused by overhead outside of the BLOCKs, is worth a look. Here are the results with the same test files as above

Audio Format	Overhead per frame @ <x> Frames per lace								
	4	8	12	16	24	32	48	64	96
MP3 @ 128 kbps	4,14	2,67	2,17	1,93	1,68	1,56	1,48	1,41	1,33
MP3 @ 192 kbps	4,25	2,79	2,30	2,06	1,81	1,75	1,61	1,54	1,47
HE-AAC @ 224 kbps	4,14	2,66	2,23	2,05	1,76	1,62	1,48	1,40	1,33
HE-AAC @ 64 kbps	4,09	2,61	2,11	1,86	1,62	1,49	1,40	1,34	1,27
LC-AAC @ 268 kbps	4,06	2,57	2,07	1,82	1,66	1,51	1,37	1,30	1,22

Now lets take the 2nd table and find out how much overhead that means in a real movie of 2 hours.

In the case of the mp3 files used in that example, one frame lasts 24ms. In the case of our LC-AAC file, one frame lasts 23,22 ms, and for the HE-AAC file we get 46,44ms.

Thus a file of 2 hours will have the following number of frames:
MP3 - 300,000
LC-AAC - 310,000
HE-AAC - 155,000.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1031
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

First, lets use the default setting of `libmatroska` (8 frames per lace) and calculate the overhead a muxing app using `libmatroska` would cause when muxing those files into a movie:

- **MP3 @ 128**: overhead = 300,000 * 2,67 = 801,000 bytes
- **MP3 @ 192**: overhead = 300,000 * 2,79 = 837,000 bytes
- **HE-AAC @ 224**: overhead = 155,000 * 2,66 = 412,300 bytes
- **LC-AAC @ 268**: overhead = 310,000 * 2,57 = 796,700 bytes

With 24 frames per lace, an MP3 block would have a duration of 576ms, an HE-AAC block even about 1 second. That means, when seeking in a file, an awkward impression of the audio being missing for a moment could occur. Thus, larger laces than 1 second are highly discouraged. Nevertheless, let's analyze the overhead in our file for laces of 24 and 96 frames each, and compare the overhead to the one caused by `libmatroska`. Here is the corresponding table:

Audio Format	Frames per lace 8	24	96
MP3 @ 128 kbps	782kB	492kB	389kB
MP3 @ 192 kbps	817kB	530kB	430kB
HE-AAC @ 224 kbps	402kB	266kB	201kB
HE-AAC @ 64 kbps	395kB	245kB	192kB
LC-AAC @ 268 kbps	778kB	502kB	369kB

As you can see, putting 24 frames in one block, compared to 8 frames, saves some overhead. However, putting 96 frames in one **BLOCK** instead of 24 saves less overhead than 24 compared to 8. As 96 frames per lace will usually cause uncomfortable seeking, it is recommended not to put more than about 24 frames in one **BLOCK**.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1032
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

7.2 Overhead of CLUSTERs

Although most of the overhead is caused by BLOCKGROUPs, the amount of overhead caused by CLUSTERs themselves is noticeable as well.

Here again the basic layout of a CLUSTER:

```
Cluster <size>
  [ CRC32 ]
  TimeCode <size> <timecode>
  [ PrevClusterSize <size> <prevsize> ]
  [ Position <size> <position> ]
  { BlockGroup }
```

First, some conventions:

- each CLUSTER has a size between 16kB and 2MB
- each CLUSTER may begin between 16MB and 4GB

As typical movie files are designed to fit on 1 or 2 CDs, or 2 or 3 of them fill one DVD, point 2 will be true for most of the clusters in typical files.

With the abovementioned restrictions on CLUSTERs, the overhead inside one Cluster will be:

- CLUSTER ID + <size>: 7 bytes
- CRC32: 6 bytes
- TIMECODE: 5 bytes
- PREVCLUSTERSIZE: 5 bytes
- POSITION: 5 bytes
- SEEKHEAD entry for CLUSTER: 17 bytes

Depending on the muxing settings, the overhead caused by one CLUSTER will be between 12 and 45 bytes.

Example: Assuming a size of 1 MB per CLUSTER, that means an overhead rate of 0,001% - 0,005%, or up to 100 kB in a file of 2GB.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1033
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

7.3 Overhead caused by Cues

Here again the layout of a CUEPOINT:

```
CuePoint <size>
  CueTime <size> <time>
  { CueTrackPosition <size>
      CueClusterPosition <size> <position>
      CueTrack <size> <track>
      [ CueBlockNumber <size> <block number> ]
  }
```

Assuming that a CUEPOINT only points into one certain track, the overhead is:

- CuePoint: 2 bytes
- CueTime: 5 bytes
- CueTrackPosition: 2 bytes
- CueClusterPosition: 6 bytes
- CueTrack: 3 bytes
- CueBlockNumber: 4 bytes

Total: `22 bytes`.

Example: Assuming that there is a CUEPOINT each 4 seconds (1 keyframe in 100 frames), this adds on overhead of `0,22 bytes / frame`

There can also be CUEPOINTs for audio tracks. In that case, as every frame will be a keyframe, the number of CUEPOINTs only depends on the muxing application. Predicting the overhead requires to know its behaviour.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1034
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8 Links

Matroska pages / software:

http://www.matroska.org
http://haali.cs.msu.ru/mkv/
http://www-user.tu-chemnitz.de/~noe/Video-Zeug/
http://de.wikipedia.org/wiki/Matroska
http://www.matroska.info/
http://ld-anime.fairreal.net/guide/jargon.matroska-en

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1035
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

AVI File Format

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1036
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Last change: December 14, 2006

Contents

1 Introduction **3**
- 1.1 Why another AVI file format documentation? . . . 3
- 1.2 Basic data structures . . . 4
  - 1.2.1 Chunks . . . 4
  - 1.2.2 Lists . . . 4
- 1.3 AVI file types . . . 5

2 Layout of an AVI file **6**
- 2.1 Headers . . . 7
  - 2.1.1 MainAVIHeader (`avih`) . . . 7
  - 2.1.2 The **Str**eam header **l**ist - general . . . 9
  - 2.1.3 The stream header list element: **strh** . . . 10
  - 2.1.4 The stream header list element: **strf** . . . 11
  - 2.1.5 The stream header list element: **indx** . . . 11
  - 2.1.6 The stream header list element: **strn** . . . 11

3 AVI Indexes **12**
- 3.1 old style index . . . 12
- 3.2 Open-DML Index . . . 13
  - 3.2.1 Upper Level Index ('Super Index') . . . 14
  - 3.2.2 The Standard Index . . . 15
- 3.3 Using the Open-DML Index . . . 17

4 The `movi` - Lists **18**

5 Audio types requiring special attention **19**
- 5.1 MP3 . . . 19
- 5.2 AC3 . . . 19
- 5.3 DTS . . . 20
- 5.4 VBR audio - general . . . 20
- 5.5 MPx VBR . . . 21
- 5.6 AAC . . . 21
- 5.7 VFR Audio - Storing Vorbis in AVI . . . 21

6 Subtitles in AVI files **22**

7 Garbage in AVI files **24**
- 7.1 Constant Bitrate . . . 24
- 7.2 Variable Bitrate . . . 24

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1037
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8 Overhead of AVI files **25**
8.1 General 25
8.2 Getting number of CHUNKS 25
8.2.1 Video 25
8.2.2 Audio 25
8.2.3 Examples 25

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1038
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1 Introduction

1.1 Why another AVI file format documentation?

Even though the AVI file format has been around for more than 10 years, there is no documentation available which does not only describe the format itself, but which also informs about issues that come from flawed demuxers and flawed decoders, and how to circumvent them.

The goal of this document is not only to explain the AVI format, as it is defined on the paper, but rather how to use it, when working with flawed muxers, flawed demuxers, and flawed decompressors.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1039
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1.2 Basic data structures

There are 2 types of atoms in AVI files:

1.2.1 Chunks

```
typedef struct {
  DWORD  dwFourCC
  DWORD  dwSize
  BYTE   data[dwSize]    // contains headers or video/audio data
} CHUNK;
```

1.2.2 Lists

```
typedef struct {
  DWORD  dwList
  DWORD  dwSize
  DWORD  dwFourCC
  BYTE   data[dwSize-4]  // contains Lists and Chunks
} LIST;
```

A chunk containing video, audio or subtitle data uses a `dwFourCC` containing 2 hexadecimal digits specifying the stream number and 2 letters specifying the data type (dc = video, wb = audio, tx = text). The values `dwFourCC` and `dwSize` have the same meaning in both of the structures:
`dwFourCC` describes the type of the chunk (for example `'hdrl'` for 'header list'), and `dwSize` contains the size of the chunk or list, including the first byte after the `dwSize` value. In the case of Lists, this includes the 4 bytes taken by `dwFourCC`!

The value of `dwList` can be `'RIFF'` ('RIFF-List') or `'LIST'` ('List').

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1040
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1.3 AVI file types

Basicly, there are 3 types of AVI files:

- **AVI 1.0** The original, old AVI file.
- **Open-DML** An extension to the AVI file format. Version 1.02 has been specified by 28/02/1996. The most important improvements are:
  - almost unlimited filesize (much more than what NTFS allows, for example)
  - overhead reduced by 33%
- **Hybride-Files**: Open-DML files that contain an additional Legacy Index for compatibility reasons. This is not an "official" word for those files, but it is describing that file type pretty well. Hybride files containing only one RIFF List can be treated as either file type.

This document describes the subset of Open-DML 1.02 file format features, as well as some additions ('hacks'), which work in common players if the proper (freely available) filters are installed. Features that work in Open-DML, but not in AVI 1.0, will be indicated to be Open-DML only.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1041
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2 Layout of an AVI file

A `RIFF-List` where `dwFourCC = 'AVI '` shall be called a 'RIFF-AVI-List', a `RIFF-List` where `dwFourCC = 'AVIX'` shall be called a 'RIFF-AVIX-List'.

Every AVI file has the following layout:

```
RIFF AVI          // mandatory
{ RIFF AVIX }     // only for Open-DML files
```

Unlike what a `uint32` suggests, the limit for the size of those lists is not 4 GB, but

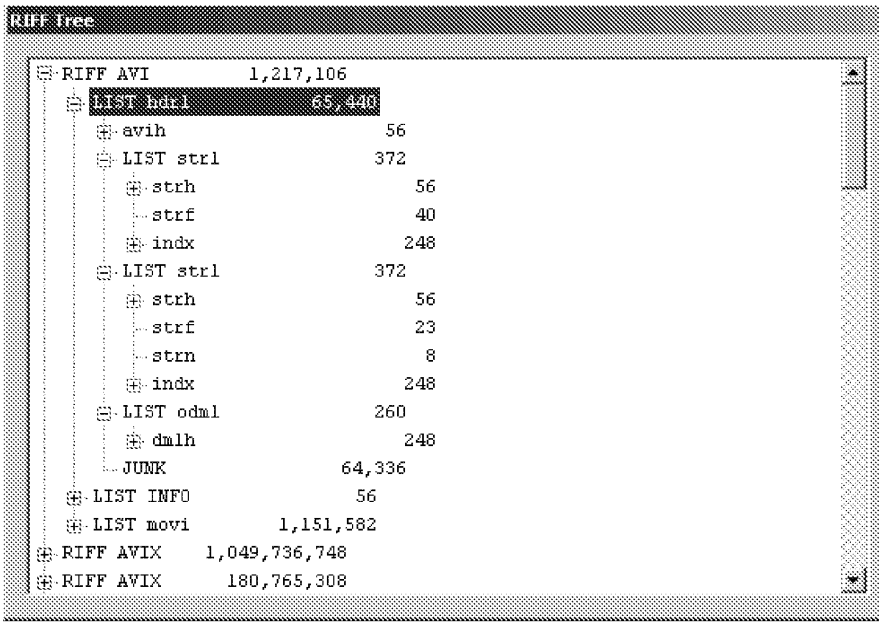
- for AVI 1.0: size(RIFF-AVI) $<$ 2 GB
- for Open-DML:
  - `size(RIFF-AVI) < 1 GB` (!!) (assumed to be 2 GB by some muxing applications, like VirtualDub!)
  - `size(RIFF-AVIX) < 2 GB`

As Windows XP insists on reading the entire first RIFF AVI list if no Legacy Index (see page 12) is found, and as that Legacy Index causes overhead, it is recommended to create RIFF-AVI-Lists as small as possible.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1042
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2.1 Headers

The header section of an AVI file looks like this:



The following sections describe the meaning of these lists and chunks.

2.1.1 MainAVIHeader (avih)

This structure is defined as follows:

```
typedef struct
{
    DWORD dwMicroSecPerFrame; // frame display rate (or 0)
    DWORD dwMaxBytesPerSec; // max. transfer rate
    DWORD dwPaddingGranularity; // pad to multiples of this
                                                // size;
    DWORD dwFlags; // the ever-present flags
    DWORD dwTotalFrames; // # frames in file
    DWORD dwInitialFrames;
    DWORD dwStreams;
    DWORD dwSuggestedBufferSize;

    DWORD dwWidth;
    DWORD dwHeight;

    DWORD dwReserved[4];
} MainAVIHeader;
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1043
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Unfortunately, those values do NOT have the meaning they seem to have when looking at their names.

- `dwMicroSecPerFrame`
  Contains the duration of one video frame in microseconds. This value can be ignored (see stream header), but shall be written correctly by any AVI writer.
  Important: Some broken programs, like *AVIFrate*, write the framerate value in the stream header, but not `dwMicroSecPerFrame`. Thus, `dwMicroSecPerFrame` should not be considered reliable!

- `dwMaxBytesPerSec`
  Highest occuring data rate within the file. That value is of no importance either. Its reliability should not be overrated.

- `dwPaddingGranularity`
  File is padded to a multiple of this

- `dwFlags`
  See below

- `dwTotalFrames`
  Contains the number of video frames in the RIFF-AVI list (it should NOT contain the total number of frames in the entire file if there are RIFF-AVIX-Lists. Some tools claiming to handle AVI files even assume this, but it definitely violates the Open-DML file format specification. Such applications are broken.) As some AVI file muxers write bad values here, this value should not be considered reliable.

- `dwInitialFrames`
  Ignore that

- `dwStreams`
  Number of streams in the file

- `dwSuggestedBufferSize`
  Size of buffer required to hold chunks of the file. The reliability of this value should not be overrated.

- `dwWidth`
  Width of video stream

- `dwHeight`
  Height of video stream

Available `Flags` for `MainAVIHeader::dwFlags`

- `AVIF_HASINDEX`
  The file has an index

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1044
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- `AVIF_MUSTUSEINDEX`
  The order in which the video and audio chunks must be replayed is determined by the index and may differ from the order in which those chunks occur in the file.
- `AVIF_ISINTERLEAVED`
  The streams are properly interleaved into each other
- `AVIF_WASCAPTUREFILE`
  The file was captured. The interleave might be weird.
- `AVIF_COPYRIGHTED`
  Ignore it
- `AVIF_TRUSTCKTYPE` (Open-DML only!)
  This flag indicates that the keyframe flags in the index are reliable. If this flag is not set in an Open-DML file, the keyframe flags could be defective without technically rendering the file invalid.

2.1.2 The Stream header list - general

There is one `strl` - List for each stream. If the number of `strl` - Lists inside the `hdrl` - List is different from `MainAVIHeader::dwStreams`, a fatal error should be reported.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1045
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2.1.3 The stream header list element: strh

```
typedef struct {
    FOURCC  fccType;
    FOURCC  fccHandler;
    DWORD   dwFlags;
    WORD    wPriority;
    WORD    wLanguage;
    DWORD   dwInitialFrames;
    DWORD   dwScale;
    DWORD   dwRate;          /* dwRate / dwScale == samples/second */
    DWORD   dwStart;
    DWORD   dwLength;        /* In units above... */
    DWORD   dwSuggestedBufferSize;
    DWORD   dwQuality;
    DWORD   dwSampleSize;
    RECT    rcFrame;
} AVIStreamHeader;
```

Again, the meaning is not always obvious.

- `fccType`
  Can be
  - `'vids'` - video
  - `'auds'` - audio
  - `'txts'` - subtitle
- `fccHandler`
  FourCC of codec to be used.
- `dwFlags`
  The following flags are defined:
  - `AVISF_DISABLED` - Stream should not be activated by default
  - `AVISF_VIDEO_PALCHANGES` - Stream is a video stream using palettes where the palette is changing during playback.
- `dwInitialFrames`
  Number of the first block of the stream that is present in the file.
- `dwRate / dwScale` –
  `samples / second` (audio) or
  `frames / second` (video).
  dwScale and dwRate should be mutually prime. Tests have shown that for example 10,000,000/400,000 instead of 25/1 results in files that don't work on some hardware MPEG4 players.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1046
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- `dwStart`
  Start time of stream. In the case of VBR audio, this value indicates the number of silent frames to be played before the stream starts.

- `dwLength`
  size of stream in units as defined in `dwRate` and `dwScale`

- `dwSuggestedBufferSize`
  Size of buffer necessary to store blocks of that stream. Can be 0 (in that case the application has to guess), but should not be 0, as Microsoft's AVI splitter does not handle this case properly in some cases (e.g. MP3-CBR in Open-DML files)

- `dwQuality`
  should indicate the quality of the stream. Not important

- `dwSampleSize`
  number of bytes of one stream atom (that should not be split any further).

2.1.4 The stream header list element: strf

The structure of the `strf` chunk depends on the media type.
Video streams use the `BITMAPINFOHEADER` structure, whereas audio streams use the `WAVEFORMATEX` structure.

2.1.5 The stream header list element: indx

This chunk contains the upper level index for the stream. See page 13.

2.1.6 The stream header list element: strn

This element contains a name for the stream. That stream name should only use plain ASCII, especially not UTF-8.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1047
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3 AVI Indexes

3.1 old style index

The index as described is the index you will find in AVI 1.0 files. It is placed after the `movi` List in the `RIFF AVI List`. The data section of the idx1 chunk has the following layout:

```
AVIINDEXENTRY  index_entry[n]

typedef struct {
    DWORD  ckid;
    DWORD  dwFlags;
    DWORD  dwChunkOffset;
    DWORD  dwChunkLength;
} AVIINDEXENTRY;
```

Those values have the following meaning:

- **ckid**
  Specifies a four-character code corresponding to the chunk ID of a data chunk in the file.

- **dwFlags**
  The following flags are defined:

  - `AVIIF_KEYFRAME`: The chunk the entry refers to is a keyframe.
  - `AVIIF_LIST`: The entry points to a list, not to a chunk.
  - `AVIIF_FIRSTPART`: Indicates this chunk needs the frames following it to be used; it cannot stand alone.
  - `AVIIF_LASTPART`: Indicates this chunk needs the frames preceding it to be used; it cannot stand alone.
  - `AVIIF_NOTIME`: The duration which is applied to the corresponding chunk is 0.

  If neither `AVIIF_FIRSTPART` nor `AVIIF_LASTPART` is set, the chunk can be used alone, in other words, it is at least one packet of the corresponding stream. This is important for storing VBR audio streams in AVI files (see chapter 5.4)

- **dwChunkOffset**
  Contains the position of the header of the corresponding Chunk.
  Warning: This can be either the absolute position in the file, or the position relatively to the first byte of the `'movi'` identificator. An AVI File parser must be able to handle both versions.

- **dwChunkLength**
  Contains the size of the corresponding chunk in bytes.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1048
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3.2 Open-DML Index

The general structure of an Open-DML-Index-Chunk is the following:

```
typedef struct _aviindex_chunk {
  FOURCC fcc;
  DWORD  cb;
  WORD   wLongsPerEntry;
  BYTE   bIndexSubType;
  BYTE   bIndexType;
  DWORD  nEntriesInUse;
  DWORD  dwChunkId;
  DWORD  dwReserved[3];
  struct _aviindex_entry {
    DWORD  adw[wLongsPerEntry];
  } aIndex [ ];
} AVIINDEXCHUNK;
```

Every subtype of Open-DML index structures is compatible to this one. The elements have the following meaning:

- `fcc`, `cb`: Chunk header, same as `dwFourCC` and `dwSize` in the `CHUNK` structure
- `wLongsPerEntry`: every `aIndex[i]` has a size of 4*`wLongsPerEntry` bytes. (the structure of each `aIndex[i]` depends on the special type of index)
- `bIndexType`, `bIndexSubType`: defines the type of the index
- `nEntriesInUse`: `aIndex[0]..aIndex[nEntriesInUse-1]` are valid
- `dwChunkId`: ID of the stream the index points into, for example `'00dc'`.
  Consequently, one such index chunk can only point to data of one and the same stream.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1049
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3.2.1 Upper Level Index ('Super Index')

The upper level index ('super index') points to other index chunks and has the following structure:

```
typedef struct _avisuperindex_chunk {
  FOURCC fcc;
  DWORD  cb;
  WORD   wLongsPerEntry;
  BYTE   bIndexSubType;
  BYTE   bIndexType;
  DWORD  nEntriesInUse;
  DWORD  dwChunkId;
  DWORD  dwReserved[3];
  struct _avisuperindex_entry {
    __int64 qwOffset;
    DWORD   dwSize;
    DWORD   dwDuration;
  } aIndex[ ];
} AVISUPERINDEX;
```

The following values are now defined more specifically:

- `bIndexType = AVI_INDEX_OF_INDEXES`
- `bIndexSubType = [ AVI_INDEX_2FIELD | 0 ]`
- `wLongsPerEntry = 4`

As you can see, the `aIndex` array now consists of 4 `DWORD`s per entry. The values have the following meaning:

- `qwOffset`: Position of the index chunk this entry points to in the file
- `dwSize`: The size of the standard or field index chunk the entry is pointing to
- `dwDuration`: The duration, measured in stream ticks as indicated in the AVI stream header. In case of video or VBR audio, that usually refers to the number of frames.
  Important:
  VirtualDub 1.4.10 and earlier versions wrote b0rked values for this member in the audio stream. Thus, an AVI parser should be able to handle files without using this value!

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1050
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3.2.2 The Standard Index

This index type contains pointers to video, audio or subtitle chunks. It also is a special form of the general Open-DML Index and looks like this:

```
  typedef struct _avistdindex_chunk {
    FOURCC  fcc;
    DWORD   cb;
    WORD    wLongsPerEntry;
    BYTE    bIndexSubType;
    BYTE    bIndexType;
    DWORD   nEntriesInUse;
    DWORD   dwChunkId;
    __int64 qwBaseOffset;
    DWORD dwReserved3;
    struct _avistdindex_entry {
      DWORD dwOffset;
      DWORD dwSize;
    } aIndex[ ];
} AVISTDINDEX;
```

- `wLongsPerEnrty`:
  As you can see easily, each `aIndex[i]` takes 8 bytes,
  so `wLongsPerEntry = 2`

- `bIndexSubType:  = 0`

- `bIndexType:= AVI_INDEX_OF_CHUNKS`

- `qwBaseOffset`:
  This value is added to each `dwOffset` value of the `AVISTDINDEX`.

- `dwOffset, dwSize`: These elements define the position (`qwBaseOffset + dwOffset`) of the data section of the corresponding `CHUNK` (NOT the chunk header!) and its length. There are `nEntriesInUse` such pairs, each one describing one video/audio frame. Note that Bit 31 of `dwSize` indicates the frametype: If this bit is set, this frame is not a keyframe.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1051
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Low-overhead mode

The Open-DML specification does not explicitly require that each "data section" the index contains an entry to be preceded by a chunk header. Thus, several frames can be put into one chunk, while having one index entry per frame. This way, only few frames have a chunk header, reducing the overhead by 50% compared to normal Open-DML files. This means, of course, that the `AVIF_MUSTUSEINDEX` flag in the main AVI header must be set, to force any parser to use the index. Files created this way will be called "low overhead AVI files".
This is not an AVI file type on its own. Any parser handling that flag, as well as the Open-DML index correctly, should be compatible to such files. Microsoft's AVI splitter, as well as VirtualDub(Mod) can handle such files without problems, without having been updated to do so.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1052
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3.3 Using the Open-DML Index

The preceding section described what the Open-DML Index looks like. This section will deal with using it.

Each stream contains an `'indx'` chunk in its stream header list (`'strl'`). This chunk is a Super Index chunk.

As each Standard Index contains one 64 bit offset and then a list of 32 bit offsets relatively to the 64 bit offset, one Standard Index chunk can only point to data within one 4 GB segment. Thus you need one Standard Index per 4 GB file size per stream.

Unfortunately, it seems that Microsoft did not read the specification properly: If a file contains more than 3 audio streams, then the Microsoft AVI Splitter will not recognize files using such large Standard Index chunks. It is required to use smaller pieces. Tests have shown that pieces with 15000 entries each are small enough to be processed correctly by Microsoft's AVI splitter.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1053
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

4 The movi - Lists

The Movi - Lists contain Video, Audio, Subtitle and (secondary) index data. Those can be grouped into rec - Lists. Example:

```
LIST movi
  LIST rec
    01wb
    01wb
    02wb
    03wb
    03wb
    03wb
    00dc
    00dc
  LIST rec
    01wb
    02wb
  LIST rec
    ...
    ...
    ix01
    ix02
    ix03
    ....
    ....
```

The following chunk header IDs are defined:

- ..wb: audio chunk
- ..dc: video chunk
- ..tx: subtitle chunk
- ix..: standard index block

Grouping chunks into rec - Lists prevents excessive seeking when using the Microsoft AVI splitter for replay, but does not allow playback on some standalone replay devices.

The maximum size of a chunk of a stream should be smaller than the corresponding dwSuggestedBufferSiz value. Otherwise, some players, especially the Microsoft AVI splitter, could malfunction.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1054
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5 Audio types requiring special attention

5.1 MP3

`wFormatTag = 0x0055`

An MP3 audio stream consists of inseparable frames. MP3 decoders should be able to handle partial frames, but it is nevertheless recommended to store entire MP3 frames in the AVI chunks.

The `strf` chunk is an `MPEGLAYER3WAVEFORMAT` structure, which is an extention to the `WAVEFORMATEX` structure:

```
typedef struct mpeglayer3waveformat_tag {
  WAVEFORMATEX  wfx;
  WORD          wID;
  DWORD         fdwFlags;
  WORD          nBlockSize;
  WORD          nFramesPerBlock;
  WORD          nCodecDelay;
} MPEGLAYER3WAVEFORMAT;
```

Important:
This is only valid for MP3 ('MPEG Layer 3'), not for MP1 or MP2 ('MPEG Layer 1 / 2').

If the MP3 stream has a variable bitrate, then you need to convince DirectShow to seek properly. See section 5.4 (page 20) for more details on VBR audio streams in AVI files.

Unfortunately, whoever came up with the idea didn't think enough about it: It is possible to create MP3 audio frames larger than 1152 bytes if the sample rate is 32 khz or less. After reading and understanding section 5.4, you'll see why such audio frames render an MP3 stream unplayable if nBlockSize is set to 1152, which is usually done for MP3. Using a larger value would resolve this issue. However, some programs read an MP3 stream as VBR if and only if this value is exactly 1152. In other words, low sample rates in combination with high bitrates are a problem for MP3 VBR streams in AVI files.

5.2 AC3

`wFormatTag = 0x2000`

Muxing AC3 into AVI is far more problematic than most other audio formats. The reason is that a lot of decoders (software as well as hardware) are severely b0rked.

An AC3 stream consists, like MP3, of individual, inseparable frames. It is required that any audio chunk of an AVI file contains a few (complete!) AC3 frames. Otherwise, some AC3 decoders will miscalculate the duration of a chunk. As the audio stream is considered the master stream for playback in DirectShow, this miscalculation will lead to jerky video playback.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1055
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Theoretically, chunks containing one AC3 frame are valid, but there are hardware decoders which won't work with such streams. If you place more than 6 AC3 frames into one chunk, you might get increased playback speed. **Thus, the recommendation is to place between 2 and 5 AC3 frames into one AVI chunk**.

5.3 DTS

`wFormatTag = 0x2001`

It seems to work out to place between 2 and 20 DTS frames in one AVI chunk. I have not yet tried higher values on my own. If you have a hardware DTS decoder, please do some testing and report back what values work and which ones don't. Just like with AC3, do not split up any DTS frames.

5.4 VBR audio - general

Just like the video stream of virtually every AVI file does have a variable bitrate and a constant frame duration (e.g. 40ms), an audio stream of an AVI file can also have variable bitrate, if the frame duration is constant. This means, placing Vorbis audio in an AVI file will NOT result in anything playable using the method described below.

As the goal is to make DirectShow seek in the stream like in a video stream, it is required to use some of the stream header values in the same way they are used with video streams. That means:

- `dwRate` contains the sample rate
- `dwScale` contains the number of samples in one audio frame
- `WAVEFORMATEX::nBlockAlign` must have a value at least as big as the largest frame of that stream, measured in bytes.
- `dwSampleSize` = 0
- Each chunk contains entire stream packets and is therefore marked as such. Neither `AVIIF_LASTPART` nor `AVIIF_FIRSTPART` is set for any chunk. This way, DirectShow assumes each chunk to contain a packet of the duration indicated in the stream headers.

This way, you can ensure that the M$ AVI Splitter will believe that the duration of every chunk is
`[roundup(chunk size / nBlockAlign)] * dwScale / dwRate`

You can, of course, create units larger than one frame to decrease overhead (independantly from "low overhead muxing" as described on page 16).
However, if a stream is VBR, it is required that every chunk (in the case of "normal" files) or every piece of audio data the index points to (in the case of "low overhead AVI files") contains the same number of samples. The `dwScale` value needs to be set accordingly.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1056
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5.5 MPx VBR

- MPEG 1 Layer 3: 1152 samples per frame
- MPEG 2 Layer 3: 576 samples per frame
- MPEG 1/2 Layer 2: Strongly discouraged from being used, because the default MPEG Layer 2 decoder present on Windows systems does not recognise MPEG Layer 2 streams when using VBR headers.

5.6 AAC

For AAC, one RAW AAC frame usually spans over 1024 samples. However, depending on the source container (e.g. ADTS), it is theoretically possible that you are not able to extract packets of equal duration from your source file. In this case, it is highly recommended not to mux the AAC stream into AVI, but report a fatal error instead.

AAC and HE-AAC require private data in the corresponding `WAVEFORMATEX` structures. See the source code of AVI-Mux GUI (`FillASI.cpp`) for details.

5.7 VFR Audio - Storing Vorbis in AVI

Generally, the AVI file format does not support variable framerate streams. However, such streams can nevertheless be stored in AVI files when it is possible to create AVI chunks of constant duration, so that the stream can be treated as a normal VBR stream. There are basicly 2 possible ways to do that:

- Building larger "macro frames" of constant duration and placing one such macro frame into one chunk. This requires a decoder that is able to decode a sequence of independant frames at once, a decoder which does not require that only one single frame at once be transmitted. The currently available Vorbis decoders do no meet this requirement.
- The duration of one chunk is set to a very small value, several empty frames are added after one that is supposed to be long. This, however, causes a lot of overhead.

`FFMPEG` is using the second idea. However, ffmpeg is increasing the overhead even more, as it stores frame headers for each of the padding chunks, wasting 8 bytes each time, instead of only storing one of them physically, adding an index entry everytime such a chunk is required, and setting `AVIF_MUSTUSEINDEX`. For such streams, the following ID is used: `wFormatTag = 0x566F`. The Vorbis initialization packets (the first 3 packets of an OGG/Vorbis file) are stored as private data in the `strf` chunk. Each vorbis initialization packet is stored the following way:

```
big_endian_int16  size
char              data[size];
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1057
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

6 Subtitles in AVI files

This section explains how to store subtitles in AVI files, so that VSFilter can be used to load and select subtitles.

One subtitle stream is stored in one single chunk. That chunk contains header data, followed by an entire SRT or SSA file. If that file is using UTF-8 encoding, the BOM should be included as well. The header is defined as follows:

```
char[4];  // 'GAB2'
BYTE  0x00;
WORD  0x02;              // unicode
DWORD dwSize_name;       // length of stream name in bytes
char  name[dwSize_name]; // zero-terminated subtitle stream
                            name encoded in UTF-16
WORD  0x04;
DWORD dwSize;            // size of SRT/SSA text file
char  data[dwSize];      // entire SRT/SSA file
```

Stream header chunk

```
typedef struct {
    FOURCC  fccType;    // "txts"
    FOURCC  fccHandler; // 00 00 00 00
    DWORD   dwFlags;
    WORD    wPriority;
    WORD    wLanguage;
    DWORD   dwInitialFrames;
    DWORD   dwScale;
    DWORD   dwRate;       // dwRate / dwScale == duration in seconds
    DWORD   dwStart;
    DWORD   dwLength;     // In units above..., should be 1
    DWORD   dwSuggestedBufferSize;
    DWORD   dwQuality;
    DWORD   dwSampleSize; // = 0 -> treated as VBR
    RECT    rcFrame;      // 0, 0, 0, 0
} AVIStreamHeader;
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1058
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Stream format chunk

This chunk has a size of 0.

Stream name chunk

The `strn` chunk is ignored by *VSfilter*, so there is no need to read or write it.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1059
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

7 Garbage in AVI files

One way to set a delay in AVI files is using the `AVIStreamHeader::dwStart` value (see page 10). However, not every player reads this value, so that such files would not properly play on them. Some applications, like *VirtualDub* and derivates (*NanDub, VirtualDubMod*) add some data to the beginning of streams to be delayed.

Unfortunately, those applications don't care about the data format, but rather pad with zeros. Those zeros are refered to as 'garbage' in this section. The scope is to explain how to read such files, which requires that the duration of the garbage section be determined.

It is of course required that the beginning of the valid data be found, for example in the case of MP3, AC3 or DTS by looking for frame headers. Formats which don't use frame headers that can be detected easily, like AAC, will result in broken streams if zeros are added to the beginning.

7.1 Constant Bitrate

If the bitrate of a stream is constant, the duration of the garbage section is simply `garbage_length * data_rate`.

7.2 Variable Bitrate

In this case, the duration must be retrieved chunk-wise:

As described on page 20, the duration of one chunk is
`roundup(size_of_chunk / strf::nBlockAlign) * duration_per_frame`.

Consequently, if the beginning of the valid data is `the m-th byte` in chunk `n`, the duration of the garbage section is

```
sum[i=0..n-1](duration(chunk[i])) + roundup(m/nBlockAlign)
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1060
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8 Overhead of AVI files

This section describes how to predict the overhead of an AVI file before muxing. Note: In the case of low overhead AVI files, the wording in this section is not applicable. Basicly, one video/audio frame causes about 8-9 bytes of overhead in low overhead AVI files.

8.1 General

The overhead of AVI files depends on the number of `CHUNK`s in the file. Other structures has only very little influence on the total overhead. Each `CHUNK` causes the following amount of overhead:

- 8 Bytes chunk header (all avi types)
- 16 Bytes for entry in Legacy Index (see page 12) (AVI 1.0 and the RIFF-AVI-List of Hybride files)
- 8 Bytes for entry in Standard Index (see page 15) (Open-DML)

That means, each `CHUNK` causes an amount of overhead of 16, 24 or 32 bytes.

8.2 Getting number of `CHUNKS`

8.2.1 Video

The easier part is the video stream: Each video frame takes one `CHUNK`.

8.2.2 Audio

The number of chunks for an audio stream depends on its format and packing. For specific formats, where very special packing is required or considered normal (see page 19), the overhead can be calculated easily from the settings. Otherwise, more precise information on the muxing settings is needed.

8.2.3 Examples

Video: 3 hours, 25 fps (= 3,600,000 / 40ms = 90,000 frames per hour)
Audio: 2x MP3-VBR (with 1 frame per `CHUNK` and 24 ms per frame)
Audio: 2x AC3 (with 4 frames per `CHUNK` and 32ms per frame)
-> Video: 270,000 `CHUNK`s
-> Audio: 2*150,000 + 2*3*28,125 = 468,750 `CHUNK`s
-> Sum = 738,750 `CHUNK`s

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1061
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Definitions

Alexander Noé, April 11, 2006

back to main page

Definitions

chunk

A chunk is the smallest piece of data in an AVI file. A chunk can e.g. contain a video frame.
structure:

```
typedef struct {
DWORD dwFourCC;
DWORD dwLength;
BYTE bData[dwLength];
} CHUNK;
```

CRC check sum

A checksum that allows the detection (but not correction!) of errors in data read for example from a disc. A typical CRC checksum is a 32 bit value.

deltaframe

A frame which, unlike keyframes, only contains the differences to the previous frame. A deltaframe can only be decompressed if the previous frame has already been decompressed.

dropped frame

A frame that doesn't contain any data. When replaying a dropped frame, the previous frame is displayed again. This technique is used in AVI files to make one frame being displayed longer than another one, such as in 24fps/30fps hybrid files, which are stored at a framerate of 120 fps, with 3 or 4 dropped frames between 2 real frames.

dwDuration

The super index contains for every standard index it points to the amount of time it takes to replay all data that standard index points to. For Open-DML files created with VirtualDub up to 1.4.10 over 2 GB, the dwDuration values of the audio stream are corrupt.

frame header

MP3, AC3 and DTS streams contain some information about the frame (size, bitrate, samping rate etc) in the first bytes of each frame. AVI-Mux GUI doesn't support MP3-VBR / DTS streams with bad frame headers. Broken AC3 stream headers will not be copied either. If a broken AC3 stream header is encountered, the next valid one will be looked for. Any data between will be omitted.

Index

An index contains information about the position of video-, audio- or other index data. For AVI files, there are several types of indexes.

legacy index

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1062
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

This index is found in normal AVI files (called AVI 1.0 sometimes). It is always stored in one piece and points to video and audio data.

super index

The super index can only be found in Open-DML files. It is the upper level index of the hierarchic Open-DML AVI file index. It points to standard- or field index data. A super index shouldn't point to other super indexes

standard index

This one is also only found in Open-DML files. It is the lower level index of the hierarchic Open-DML AVI file index. It points to video- and audio data, using less space for one entry than the legacy index.

keyframe

A keyframe contains, unlike a deltaframe, a complete frame. In AVI terms, a keyframe can be seeked to very easily. In MPEG4-terms, it is not that easy: A B-frame succeeding such a keyframe could reference another frame preceding this keyframe. Then, the frame is still a keyframe, but not a so-called IDR-Frame. When storing MPEG4 in AVIs, reasonable encoders or muxers should only label IDR-frames as keyframes

Mode 2 Form 2

This is a possibility to store less error correction data to a CD and use the free space for own purposes. A sector can then contain 2324 bytes of user data instead of 2048. This is used for VCD and S-VCD for example. A "700 MB"-CD-R can then contain 798 MB without overburning. If you want to do this with AVI files, then please visit the homepage of Mode2CDMaker and read its readme-file carefully.

Open-DML

An extension to the AVI file format that allows the creation of AVI files larger than 2 GB. Open-DML files usually contain a hierarchic index structure with 2 levels (super- and standard indexes), and an additional legacy index is allowed. Read this to learn more or get the entire specification here.

Warning: Many hardware divX Players do not want to read Open-DML files!

rec-list

A special structure in an AVI file that contains lots of video and audio data. Such a structure is read in one piece by all players I know, so that larger blocks are read from the CD and less seek operations are necessary.

Warning: Some or all KISS players won't read files using rec lists. Disable rec lists if you intend to play your files on KISS players!

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1063
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Documentation

Widevine Content Protection

Ooyala provides support for Widevine to meet the content protection requirements for high-quality content online in on-demand and live streaming formats. Widevine's multi-platform DRM provides the capability to license, securely distribute and protect playback of multimedia content on any consumer device. Content owners and digital media providers can use Widevine's solutions to ensure revenue generating services keep flowing to whatever device consumers desire.

To give you the capacity to protect your content using Ooyala and Widevine, you need to understand how Widevine works and how to use Widevine with your iOS, Android or Connected TV device. This document describes how to do this.

Widevine is part of a set of comprehensive content protection features that work together to provide you with the ability to secure your content. These features include:

- User Authentication through token-based options such as the Ooyala Secure Player Token. For information about setting up and using this feature, see Ooyala Player Token.
- Content Authorization through mechanisms such as Widevine. Information about Widevine is provided in this document.
- Authorization API which is a mechanism that handles all authorization requests. For more information about the Authorization API, see Player Authorization API.

The content protection that Ooyala offers work standalone or in combination to provide multiple levels of content protection. Ooyala enables you to combine these features to create your content protection strategy. To learn about additional content protection features (such as Adobe Pass), contact Sales, your Customer Success Manager, or Technical Support.

Supported Platforms and Formats

If you want to have content that is DRM-protected with Widevine, you need to use the supported platforms

formats appropriate for iOS Apps, Android Apps, and Connected TVs.

Platform	Supported Formats	Notes
Android 2.x	MP4	Android 2x does not support live streaming.
Android 3.0+	MP4, WVM (VOD)HLS (Live)	WVM is Widevine ABR.
iOS	WVM (VOD) HLS (Live)	Although MP4 is available, we should encourage customers to move away from using this format.
ConnectedTVs and STBsSamsung 2010+LG2011+	WVM (VOD) HLS (Live)What the TV supports	For apps that support WVM or HLS.Note that other Connected TVs and STBs may work but have not yet been tested by Ooyala.

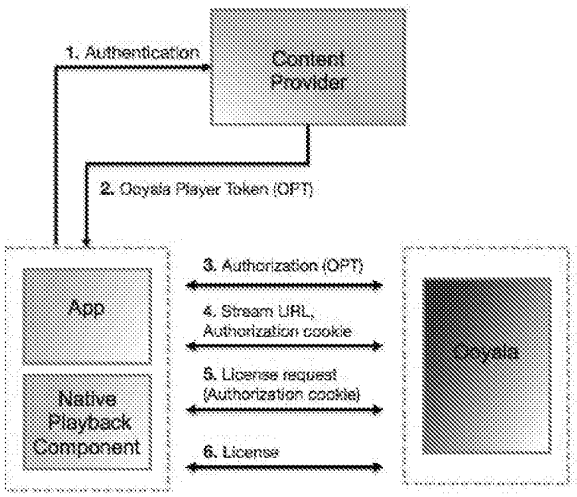
Ooyala and Widevine DRM Workflow

Customers who want Widevine need to work with Ooyala Customer Success Managers or professional services to enable Widevine support. When you use Ooyala's Widevine server, the licenses exist in the Widevine cloud.

With the implementation of Widevine DRM, each content viewer needs an individual license as the content is encrypted and is useless without license. Widevine does not perform the encryption, content from the CDN is already encrypted. Widevine just provides a secure cloud for license key storage and retrieval.

If you want to integrate with Widevine, you need to understand how securing your content works with the various parts of the Ooyala content protection features. The following diagram illustrates how Widevine is related to the Ooyala Player Token (an optional but recommended user authentication feature) and the Ooyala Authorization API that handles the user authentication requests.

Widevine Content Protection Workflow



The following table describes the workflow steps for using Widevine for content protection.

Step	Action	Responsible Party	Additional Documentation
1.	The video player App, authenticates the user against the content provider.	App developers and content provider's services	Ooyala Player Token
2.	The content provider provides an Ooyala Player Token to the app that indicates the authentication status of the user.	App developers and content provider's services	Ooyala Player Token
3.	The video app makes an authorization request to Ooyala that includes the Ooyala Player Token.	App developers	Ooyala Player Token and Player Authorization API
4.	Ooyala's authorization request returns a stream url and the authorization cookie.	App developers	Player Authorization API
5.	Device native playback components will contact Ooyala for DRM licenses.	Ooyala	
6.	Ooyala provides the license to playback DRM content.	Ooyala	

Using Widevine with Ooyala SDK-based Apps

To use Widevine with our iOS and Android players you need to download and use our Native SDKs to create the client-side pages. To download the Ooyala:

- Android SDK - http://support.ooyala.com/developers/resources#mobile
- iOS SDK - http://support.ooyala.com/developers/resources#mobile

Once you have downloaded the SDK, you should following the instructions in the package for implementing Android or iOS apps. That's all you need as once you have the Widevine feature enabled, the Ooyala SDK-based apps should work seamlessly with Widevine.

Using Widevine with Connected TVs

If you want to use Widevine with connected TVs, you need to:

1. Implement Widevine in accordance with the device specific SDKs. You pre-configure your device SDK and Widevine set up according to the SDK instructions.
2. Initialize Widevine and the device SDK in the SDK-specific language (this will vary from device to device). Set up your app in accordance with the applicable platform programming guide.
3. You will make a call the Authorization API (this is a JSON RESTful API call).
4. You will make the call to get the Widevine URL or stream data.
5. You need to map the elements of the retrieved URL to the corresponding values of the Widevine fields. Plug in the values described below.

For this segment...	Provide...
video url	the stream URL
drm server	Widevine server path in the Authorization response
portal, provider, or owner id	ooyala
device id	optional, unique identifier for the device, you generate the id

Authorization Response with Widevine Stream

For an authorization response that include a widevine stream, an additional value named widevine_server_path is returned. This should be passed along with the stream url to the playback SDKs for obtaining the Widevine license and decrypting the token.


```
 {
"authorization_data":{
   "huNWp2NjoCaCKrsV_wqBdcSw9P1XmlwW":{
      ...
      "streams":[
         {
            "delivery_type":"wv_wvm",
            "url":{
               ...
            },

"widevine_server_path":"http://player.ooyala.com/sas/drm2/lvcjAxOj82_rjlIAJ6Jr8Z
ZqGP-s/huNWp2NjoCaCKrsV_wqBdcSw9P1XmlwW/widevine/ooyala"
         }
      ]
   }
},
"debug_data": ...,
"signature":"Fo6ewzq2tTrLJrFmjo5eQpeKUOoLvhSen7KLjrFU1YQ=\n"
}
```


Parent topic: Digital Rights Management

SEND FEEDBACK

Rate this article:

OOYALA

© 2014 Ooyala, Inc. • Website Privacy Policy • Terms of Service • All Rights Reserved

streamingmedia.com



HOME | EUROPE SITE | PRODUCER | SUBSCRIBE | ARTICLES | VIDEO | TOPICS | VERTICALS | RESOURCES | WEB EVENTS | CONFERENCES | ABOUT



The 2012 Encoding and Transcoding Buyers' Guide

Discover the best hardware and software for video encoding and transcoding. We spotlight solutions from Elemental, Digital Rapids, Apple, Adobe, Sorenson, Harmonic, Telestream, Wowza, and Encoding.com.

By *Jan Ozer*

For the rest of the August/September 2012 issue of Streaming Media magazine please *click here*



The streaming media encoding universe has evolved into a complex space with multiple, often intersecting markets. In this overview article, I'll identify the most critical market segments and discuss one or two key products within each market.

BOOKMARK/SHARE

Email
Print
Digg
Send
Like 11
Tweet 12
Share 0
+1 3

My criteria for product selection are unavoidably subjective; in most instances, I chose the products that I knew the best and have worked with the most, and know to be competent at their intended tasks. This doesn't mean that the products that I didn't choose aren't worthy; rather, it just means that I wasn't as familiar with them as the products that I wrote about.

Let's start with on-demand and live hardware products, and then work our way to software and cloud- based encoders.

Hardware

Hardware On-Demand Encoder: Elemental Server

With entrants from industry veterans such as Envivio, Inc.; Digital Rapids; Cisco; and others, there are lots of worthy products in this category. That said, choosing the representative product, Elemental Server, was still an obvious choice; it's the fastest H.264 encoder that I've ever tested, with very good quality. With users such as ESPN and multiple Olympic broadcasters, it's clear that I'm not the only one who thinks so.

The Linux-driven server incorporates two six-core Westmere-class Intel Xeon CPUs and two NVIDIA Corp. GPUs, and it has an MSRP of $25,950. Though operation is browser-based, there's a lot of logic built into the interface, including error checking, which helps prevent launching jobs that will fail after processing begins. Elemental Server's deinterlacing function is outstanding -- it's a key quality consideration for those distributing interlaced-source footage. Closed caption support is quite extensive, including the ability to embed subtitles or create subtitle files (SRT, SMI, and TTML files) for MPEG-2 transport streams, Smooth Streaming, Flash, and Apple iOS devices.

Server is kind of a one-trick pony; its Windows Media output is subpar, and it doesn't output VP6 or WebM. But if high-volume, high-quality H.264 encoding is your need, Elemental Server should be atop your short list.

Companies Mentioned:

Adobe
CISCO
digital rapids
ELEMENTAL
encoding.com

SEARCH STREAMINGMEDIA.COM



UPCOMING INDUSTRY CONFERENCES
Streaming Forum 2014 (London) [24-25 June 2014]
Streaming Media West 2014 [18-19 November 2014]
Streaming Media Producer Live [18-19 November 2014]
Recent Conferences
Streaming Media East 2014 [13-14 May 2014]
Content Delivery Summit 2014 [12 May 2014]
Streaming Media West 2013 [18-20 November 2013]
Streaming Media Producer Live 2013 [19-20 November 2013]
Past Conferences



POPULAR | ARTICLES | NEWS | FORUMS

TOP 10 MOST POPULAR ARTICLES

The Codecs That Make UHD Video Possible: HEVC Vs. VP9

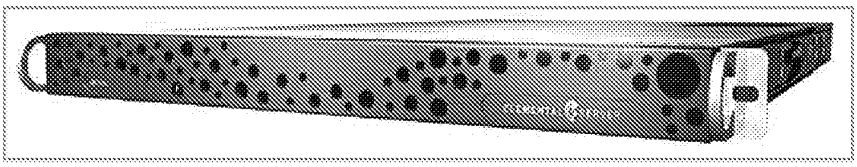
Video for the Long Haul: Exploring Backhaul Options

Streaming Media East: Branded Video Tips for Vine and Instagram

Streaming Media East: Roku Rethinking its Approach as More MVPDs Sign On

Streaming Media East: Netflix Making the Move to HEVC, but Efficiency Gains Lag

Streaming Media East: Cooperation



The Elemental Server—the one to choose for high-volume, high-quality H.264 encoding.

Hardware Live Encoder: Digital Rapids **TouchStream**

I could have gone two ways with this category: choose a big-iron, on-premise encoder for periodic or continuous broadcasts, or highlight a portable unit for live event broadcasting from the road. I chose the latter because of familiarity with the market and specifically the product that I chose, the Digital Rapids TouchStream portable encoder.

One thing all live event producers can agree upon is that less is more when it comes to equipment that you have to carry to the event and then back home. Since you can run the shoe-box-sized TouchStream via its touchscreen interface, this means that you don't need to pack a monitor or keyboard, as you would with competitive units from other vendors.

Digital Rapids offers an extensive range of models that input signals from composite SD to HD SDI and can output H.264, VP6, MPEG-2, and Windows Media Video in single stream or adaptive streams suitable for Smooth Streaming, Dynamic Streaming, and HTTP Live Streaming.

The Windows-based software is easy to use and functional, with useful adjustments for brightness, contrast, saturation and tint, special and temporal noise reduction, and audio volume controls. On the other hand, the template creation software is overly complex, but once they've create their presets, most users won't need to use this software. Most importantly, the H.264 and WMV streams I created during my testing showed uniformly good quality. TouchStream units aren't cheap, but if you're looking for a high-quality, highly portable live encoder for road productions, TouchStream is a great choice.

With our hardware categories addressed, let's move to software.



Digital Rapids TouchStream is easy to use and highly portable, and it outputs high-quality video.

Software

Bundled Software Encoders: Apple **Compressor 4**

Though Apple Compressor hasn't had a serious feature or interface update in roughly forever, it's the encoding tool of choice for most Final Cut Pro X (FCPX) editors, primarily because of cost ($49.99) and the convenience of being able to export directly from the FCPX timeline. Compressor's strong points include rock- solid compatibility with all Apple devices and relatively competitive encoding times, in part courtesy of Apple's Qmaster, which lets Compressor run multiple encoding instances on multiple core computers. Compressor also offers a very simple clustering function, which allows multiple users on a LAN to offload encoding chores to other computers on the network that have Compressor installed.

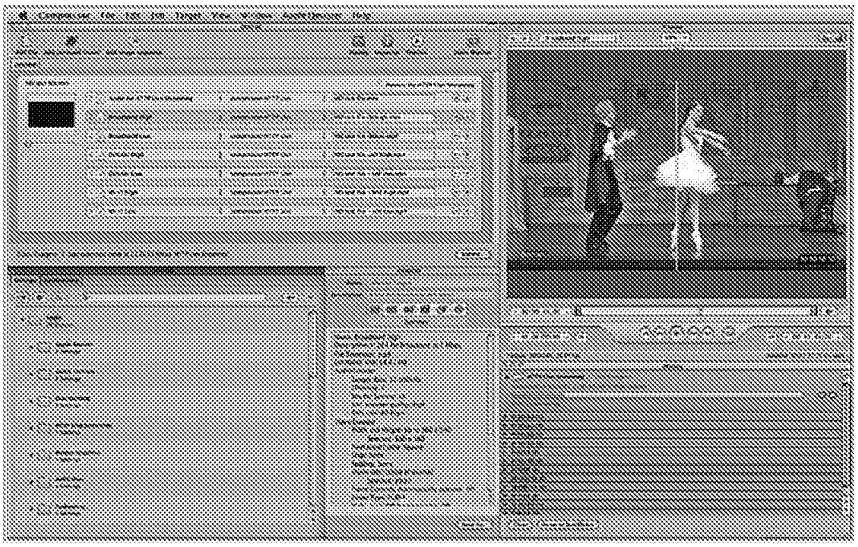
Negatives include lack of support for the H.264 High profile, which is perhaps why Compressor's quality trails that of other encoders at more aggressive data rates. Compressor also takes Apple's normal xenophobic view of formats that don't run on iOS devices, with no support for either VP6 or WMV. Also lacking is watch folder operation, limiting Compressor's ability to share encoding chores with others in a workgroup who don't have Compressor installed.

research
Download TODAY!


Streaming Media Magazine
May 2014
Subscribe
Current Issue Past Issues
- Comment: Is Fire TV Really Just Home Shopping Network in a Box?
- Colleges Need to Teach the Teachers to Produce Quality Video
- Forget 4K, Think 4D: Time Could Be a New Wrinkle in Online Video
- An Image Consultant Gives Fashion Advice to Google Glass
- Acorn TV and DramaFever: Niche Video Thrives With Foreign Flavor

Current Digital Edition

streaming media
A WORKFLOW FOR
MULTISCREEN DELIVERY

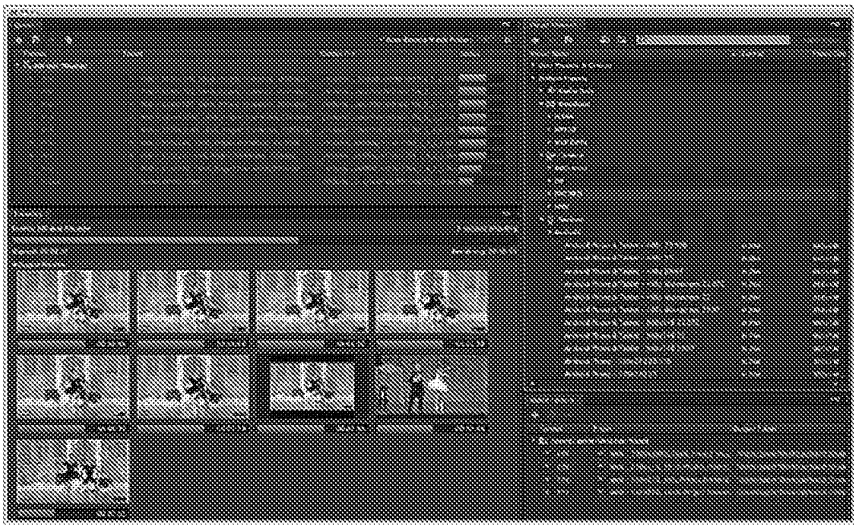


Appie Compressor s (very) familiar interface.

Adobe Media Encoder CS6

The Japanese term Kaizen refers to the practice of making many small but continuous improvements in a product over time, and it's the perfect term to describe how Adobe has improved Adobe Media Encoder (AME). Each version incorporates meaningful interface enhancements; in CS5 Adobe simplified watch folder operation and in CS6, Adobe made presets easier to find and apply. Plus CS6 brought parallel encoding, a huge benefit for those producing for adaptive streaming.

Many other features make AME the encoder of choice for Creative Suite output, including the ability to export to AME from Premiere Pro and After Effects, and the ability to directly load Premiere Pro sequences and After Effects compositions without opening the originating programs. Format support is also extensive with multiple presets for Flash, Android, and iOS output, with VP6 support on the Windows and Mac platforms, and Windows Media output on Windows installs. H.264 quality is also very competitive, with watch folders available for shared use in a workgroup.

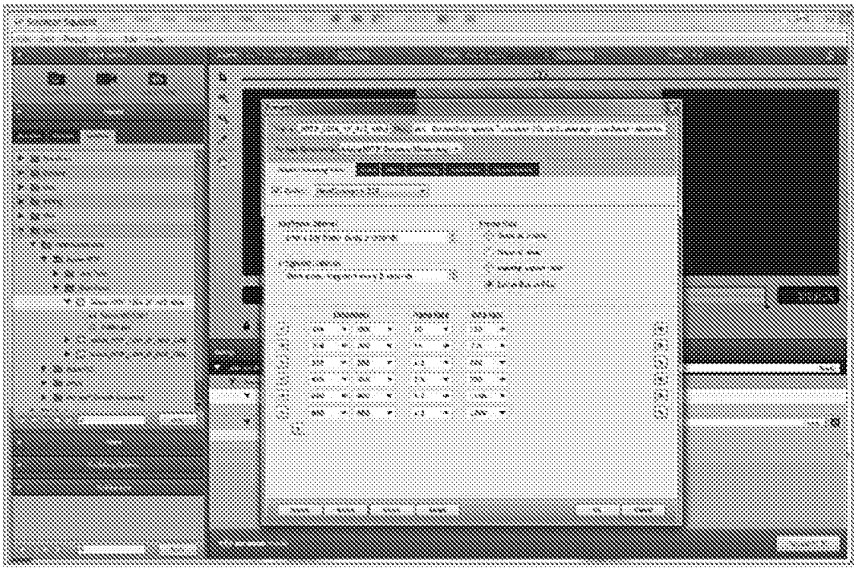


Adobe Media Encoder CS6 brought improved preset usability and parallel encoding.

Desktop Encoder: Sorenson Squeeze

Sorenson Squeeze is another product that just keeps getting better. With version 8.0, Sorenson added support for the x264 codec, which delivers better quality than the older MainConcept codec, which is still included. The newest version, Squeeze 8.5, adds accelerated encoding for single files plus unparalleled support for adaptive streaming, including presets for HTTP Live Streaming, HTTP Dynamic Streaming, Smooth Streaming, and DASH, as well as a foolproof interface that ensures consistent key frame intervals and audio parameters in all files in an adaptive group.

Other Sorenson products also add significant value to Squeeze. For example, Squeeze users can send encoding jobs to installations of Squeeze Server for high volume, industrial strength encoding, with client review and approval and video publishing easily accessible in Sorenson 360. Squeeze also outputs all relevant streaming formats, including WebM, VP6, and Windows Media and intermediate formats such as Avid DNxHD and ProRes. For most streaming formats, Squeeze offers an extensive range of configuration options that are ideal for compression experts, with presets to simplify encoding for nontechnical users.



Sorenson Squeeze s adaptive streaming preset for HTTP Dynamic Streaming.

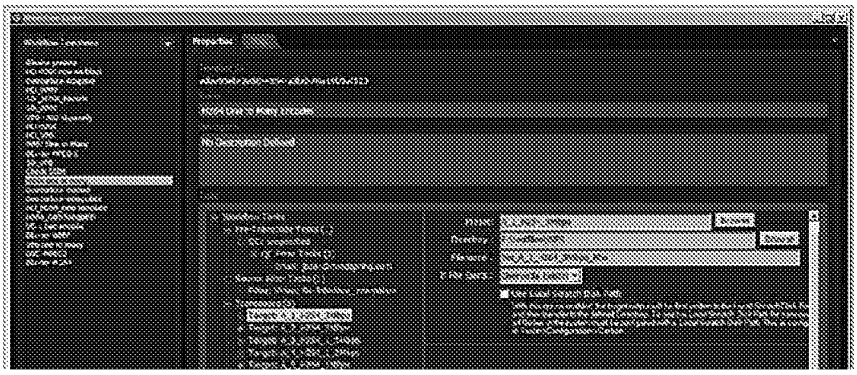
Enterprise Encoder: Harmonic's Workflow System

Harmonic, Inc.'s Workflow System (WFS) is a soup-to- nuts, on-demand encoder that will retrieve your files from a variety of remote locations; encode them to a range of streaming, intermediate, and playout formats; check the files for quality and conformity and deliver them over FTP, HTTP and CIFS. As the name suggests, WFS is a task-based engine that controls multiple other Harmonic modules, including ProMedia Carbon (formerly Carbon Coder) and the Rhozet Quality Control System (QCS).

Carbon has long been my go-to encoder for high-volume, high-quality, multiple-format encoding, though its performance has slipped recently compared to its peers. However, the ProMedia Xpress software that I recently beta tested was the fastest software-only encoder I've seen by a wide margin; it will likely restore Carbon to the top spot in the near term.

Harmonic's QCS program has all the bells and whistles of its competitors, and it can be used both pre-encode for file triage and quality control checks and postencode. However, what truly separates WFS from its closest competitors is sheer usability. While perhaps not as programmable as Telestream Vantage, for example, it's much simpler to use, and it's much more accessible for nontechnical users.

Of all the categories discussed here, the enterprise encoder market should see the most upheaval over the next few months, as virtually every vendor in the space, including Digital Rapids, Harmonic, Sorenson, and Telestream, release significant updates or new enterprise- oriented products. So stay tuned.





Harmonic WFS: Creating a workflow with pre-encode and postencode quality control checks.

Live Streaming Encoding Software: Telestream Wirecast

Usually, it only takes about 5 minutes into your first video-only webcast before you start thinking, "Gosh, this would look really great with titles for the speakers," and "What the heck am I going to broadcast during breaks or intermissions?" Well, you can spend thousands on hardware that can incorporate these elements, or you can buy Telestream Wirecast, which starts at $449 and lets you add titles to your videos, play back hard disk- based videos and much, much more, including switching between multiple cameras.

For example, you can broadcast the screen of any computer on the same network as the Wirecast capture station, a great feature for software training or presentations. Wirecast also includes live chroma key capabilities which the Pro version of Wirecast ($995) complements with 3D virtual sets. Broadcasting a sporting event? Wirecast Pro also includes an integrated scoreboard. Available for both Macintosh and Windows, both versions can produce files for Flash and QuickTime broadcasting, while the Windows version can also output Windows Media Video.



Wirecast let me add simple titles to this webcast and post the agenda slate during the intermissions.

Server-Based Software: Wowza Transcoder

Wowza Media Server was one of the first products to transmux incoming H.264 streams into formats suitable for delivery to a range of devices. Wowza Transcoder takes this one step further, transcoding a single incoming high-quality stream into multiple output streams for adaptive streaming, with minimal latency. This solves two very major problems for live streaming producers, encoding density at the event and limited outbound bandwidth from the event.

Specifically, assume that an event producer wanted to distribute six adaptive streams at 2.5Mbps, 1.8Mbps, 1.4Mbps, 1Mbps, 600Kbps, and 300Kbps. This would typically require at least two encoding stations on site, which is expensive and adds to the transportation and setup burden. Getting the streams out of the building would also require 7.7Mbps of outbound bandwidth for the streams, plus some overhead.

With Wowza Transcoder, event producers can broadcast a single 2.5Mbps stream to the remote server, where it can be re-encoded into the planned adaptive streams. Server capacity can be spun up as required, and the outbound bandwidth requirements are cut by 66%. Transcoded streams can be delivered as adaptive bitrate files using Flash RTMP/HTTP Dynamic Streaming, Apple HTTP Live Streaming, and Silverlight Smooth Streaming, or as single bitrate stream over any Wowza Media Server

3-supported protocols.



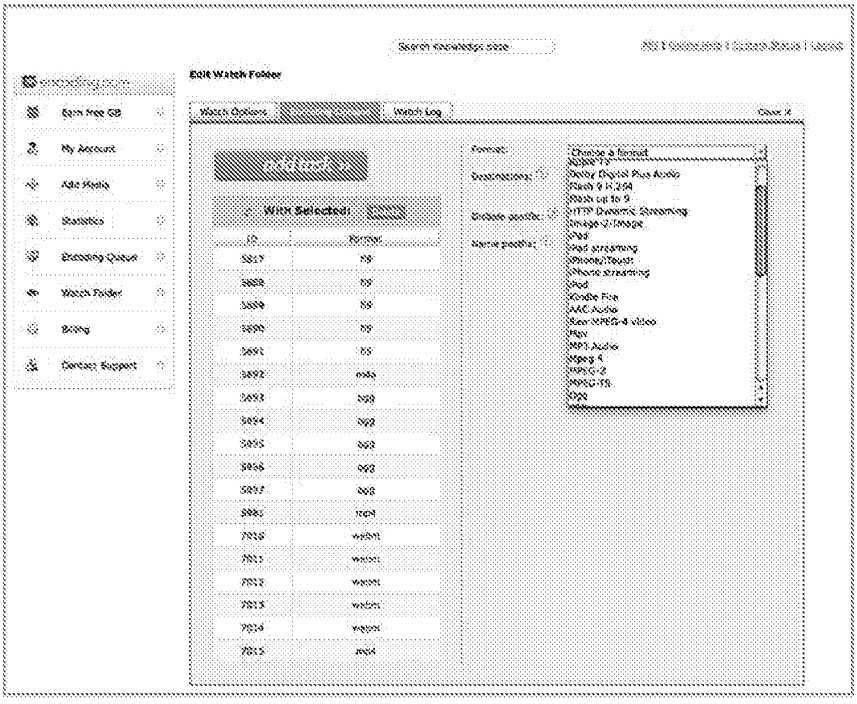
Wowza Transcoder lets event producers broadcast a single stream to the remote server, where it can be re-encoded into multiple streams.

Cloud Encoding: Encoding.com

Cloud encoding is the least familiar category to most readers and -- to be honest -- to me as well. I have worked with several clients who use Encoding.com, my cloud encoding platform of choice, so I am familiar with its workflow and encoding tools.

In addition to an application programming interface (API), Encoding.com works with watch folders, so you don't have to hire a programmer to start encoding. The service supports most relevant codecs and adaptive streaming formats with extensive presets presented in a logical interface for easy customization. And you can upload a file to check out the service, though with a 100MB limitation, you really can't test quality.

Speaking of quality, how does Encoding.com's quality and performance compare to other services such as Zencoder? I honestly don't know because I haven't run any comparative tests. As upload bandwidths increase and the velocity of relevant formats continues to accelerate, cloud encoding will gain in importance and will be a market to which we allocate more reviewing resources over the next few months.



Encoding.com supports most relevant codecs and adaptive streaming formats.

[This article appears as "Buyer's Guide: Encoding and Transcoding" in the August/September issue of Streaming Media magazine, which will reach subscribers' mailboxes and go online as a full digital

issue soon.]

Write a comment...

Comment using...



Todd Stark · Works at Stark Video, Inc

I am really struggling with converting 1920x1080 60P NTSC footage to Youtube. Its a bit jerky at times (there is lots of movement) I have tried Compressor, Media Encode, Squeeze and Episode only to see similar results. We convert down to .264 . We tried 720P and 1080P and even convert down from 60fps to 30fps before converting to .264 and same results. I am now considering trying Carbon Coder. Anyone have any ideas for me. At this time, I think its Youtubes fault and Im thinging of switching to the Pro Vimeo for its HD quality. Thanks for any replys in advance

Reply · Like · March 2 at 7:08am



Adam Urbina · Promoter at Headrushlive.com/drum&bass

Go try some aPowersoft. I tend to use this when I cant play .m4a files. It also works in other ways, which I wont divulge in public domain.

Reply · Like · May 5 at 2:42pm



Chris Thompson · Works at Mercy TV

Apple compress 4..pretty good

Reply · Like · January 10 at 8:50am



Sharon Malkiel · Tel Aviv University | אוניברסיטת תל-אביב

AVS Video Converter - Semi professional , but amazing quality.

Reply · Like · August 27, 2012 at 8:34am



Ludovic Bostral · Co-Founder & CTO at AFROSTREAM

What about cisco transcoder (ex-inlet?)

Reply · Like · August 22, 2012 at 6:00am



Jason Thomas · Works at Brazoria Telephone

Cisco is a want to be player. it's a 1990's data company that is trying to play me too, in every other arena

Reply · Like · January 1, 2013 at 6:15pm



Christian Avalos · Top Commenter · Director at GRUPO Z

I think Cisco Media Processor (Ex- Inlet) should have the best quality of them all only second by Digital Rapids. Elemental is an amazing piece of hardware but I'm not a fan of GPU assisted encoding/transcoding for quality issues below 1,2 Megabit/s.

Reply · Like · 3 · February 7, 2013 at 11:51am

Facebook social plugin

Related Articles

Featured Articles: Encoding for YouTube: How to Get the Best Results
Yes, there is a way to produce great-looking YouTube videos every time. Read on for expert advice and test results.

Sponsored Articles: Telestream Wirecast Makes Live Video Flexible and Fun
Professional-level live video streaming doesn't have to be complicated. Wirecast works with a variety of partners, and it's a pleasure to use.

Online Video News: Brightcove Acquires Zencoder
Brightcove buys the Bay Area cloud computing company for $30 million; will allow it to remain independent and even power competing services.

Featured Articles: How to Choose an Enterprise-Class Video Encoder
Hardware? Software? A workflow system? What are the advantages to each? For those not sure where to start, look here first.

SPONSORED LINKS

Looking for Trusted Suppliers? Check out the Industry Directory here

INSIDE STREAMINGMEDIA.COM

Blog
SM Magazine
SM Digital Edition
Streaming Media App
Streaming Media TV
Conference Videos
Video Tutorials
Readers' Choice
Research Reports
Press Releases
Web Events
About Us
Newsletters
Industry News
SM White Papers
Industry White Papers
Sponsored Articles
Forums
Discussion Lists
Industry Directory
Industry Jobs
Contact Us
What is Streaming

RESEARCH CENTERS

TOPICS
Mobile Video
Music/Radio
Online Video Platforms
OTT Video
Video Advertising
Video Content Protection/DRM
Video Creation/Production
Video Encoding/Transcoding
Video Formats/Players/Codecs
Video Hosting/Delivery
Video Legal Issues
Video SEO

VERTICALS
Education Video
Enterprise Video
Government Video
Media/Entertainment Video
Online Video Industry

OTHER IT SITES

Streaming Media Europe
Streaming Media Producer
destinationCRM
EContent
Faulkner Information Services
Information Today
Information Today Europe
ITIResearch.com
KMWorld
OnlineVideo.net
Smart Customer Service
Speech Technology

© 2005 - 2016, StreamingMedia.com, an Information Today, Inc. (ITI) company

{PRIVACY/COOKIES POLICY}

Informational
Internet-Draft
Intended status: Informational
Expires: April 18, 2013

R. Pantos, Ed.
W. May
Apple Inc.

October 15, 2012

HTTP Live Streaming
draft-pantos-http-live-streaming-10

Abstract

This document describes a protocol for transferring unbounded streams of multimedia data. It specifies the data format of the files and the actions to be taken by the server (sender) and the clients (receivers) of the streams. It describes version 5 of this protocol.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 18, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

This Informational Internet Draft is submitted as an RFC Editor

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1079
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Contribution and/or non-IETF Document (not as a Contribution, IETF Contribution, nor IETF Document) in accordance with BCP 78 and BCP 79.

Table of Contents

1. Introduction 4
2. Summary 4
3. The Playlist file 4
3.1. Introduction 4
3.2. Attribute Lists 5
3.3. Standard Tags 6
3.3.1. EXTM3U 6
3.3.2. EXTINF 6
3.4. New Tags 7
3.4.1. EXT-X-BYTERANGE 7
3.4.2. EXT-X-TARGETDURATION 7
3.4.3. EXT-X-MEDIA-SEQUENCE 8
3.4.4. EXT-X-KEY 8
3.4.5. EXT-X-PROGRAM-DATE-TIME 10
3.4.6. EXT-X-ALLOW-CACHE 10
3.4.7. EXT-X-PLAYLIST-TYPE 10
3.4.8. EXT-X-ENDLIST 11
3.4.9. EXT-X-MEDIA 11
3.4.9.1. Rendition Groups 13
3.4.10. EXT-X-STREAM-INF 14
3.4.10.1. Alternative Renditions 15
3.4.11. EXT-X-DISCONTINUITY 16
3.4.12. EXT-X-I-FRAMES-ONLY 16
3.4.13. EXT-X-MAP 17
3.4.14. EXT-X-I-FRAME-STREAM-INF 17
3.4.15. EXT-X-VERSION 18
4. Media segments 19
5. Key files 20
5.1. Introduction 20
5.2. IV for [AES_128] 20
6. Client/Server Actions 20
6.1. Introduction 20
6.2. Server Process 21
6.2.1. Introduction 21
6.2.2. Sliding Window Playlists 23
6.2.3. Encrypting media segments 23
6.2.4. Providing variant streams 24
6.3. Client Process 25
6.3.1. Introduction 25
6.3.2. Loading the Playlist file 25

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1080
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

6.3.3. Playing the Playlist file 26
6.3.4. Reloading the Playlist file 26
6.3.5. Determining the next segment to load 27
6.3.6. Decrypting encrypted media segments 28
7. Protocol version compatibility 28
8. Examples 29
8.1. Introduction 29
8.2. Simple Playlist file 29
8.3. Sliding Window Playlist, using HTTPS 30
8.4. Playlist file with encrypted media segments 30
8.5. Variant Playlist file 30
8.6. Variant Playlist with I-Frames 31
8.7. Variant Playlist with Alternative audio 31
8.8. Variant Playlist with Alternative video 31
9. Contributors 32
10. IANA Considerations 32
11. Security Considerations 33
12. References 34
12.1. Normative References 34
12.2. Informative References 36
Authors' Addresses 36

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1081
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1. Introduction

This document describes a protocol for transferring unbounded streams of multimedia data. The protocol supports the encryption of media data and the provision of alternate versions (e.g. bitrates) of a stream. Media data can be transferred soon after it is created, allowing it to be played in near real-time. Data is usually carried over HTTP [RFC2616].

External references that describe related standards such as HTTP are listed in Section 11.

2. Summary

A multimedia presentation is specified by a URI [RFC3986] to a Playlist file, which is an ordered list of media URIs and informational tags. The URIs and their associated tags specify a series of media segments.

To play the stream, the client first obtains the Playlist file and then obtains and plays each media segment in the Playlist. It reloads the Playlist file as described in this document to discover additional segments.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. The Playlist file

3.1. Introduction

Playlists MUST be Extended M3U Playlist files [M3U]. This document extends the M3U file format further by defining additional tags.

An M3U Playlist is a text file that consists of individual lines. Lines are terminated by either a single LF character or a CR character followed by an LF character. Each line is a URI, blank, or starts with the character '#'. Blank lines are ignored. White space MUST NOT be present, except for elements in which it is explicitly specified.

A URI line identifies a media segment or a variant Playlist file (see Section 3.4.10). Each media segment is specified by a media URI and the tags that apply to it.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1082
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Lines that start with the character '#' are either comments or tags.

Tags begin with #EXT. All other lines that begin with '#' are comments and SHOULD be ignored.

A URI in a Playlist, whether it is a URI line or part of a tag, MAY be relative. Relative URIs MUST be resolved against the URI of the Playlist file that contains it.

The duration of a Playlist file is the sum of the durations of the media segments within it.

Playlist files whose names end in .m3u8 and/or have the HTTP Content-Type "application/vnd.apple.mpegurl" are encoded in UTF-8 [RFC3629]. Files whose names end with .m3u and/or have the HTTP Content-Type [RFC2616] "audio/mpegurl" are encoded in US-ASCII [US_ASCII].

Playlist files MUST have names that end in .m3u8 and/or have the Content-Type "application/vnd.apple.mpegurl" (if transferred over HTTP), or have names that end in .m3u and/or have the HTTP Content-Type type "audio/mpegurl" (for compatibility).

3.2. Attribute Lists

Certain extended M3U tags have values which are Attribute Lists. An Attribute List is a comma-separated list of attribute/value pairs with no whitespace.

An attribute/value pair has the following syntax:

AttributeName=AttributeValue

An AttributeName is an unquoted string containing characters from the set [A..Z] and '-'.

An AttributeValue is one of the following:

- decimal-integer: an unquoted string of characters from the set [0..9] expressing an integer in base-10 arithmetic.

- hexadecimal-integer: an unquoted string of characters from the set [0..9] and [A..F] that is prefixed with 0x or 0X and which expresses an integer in base-16 arithmetic.

- decimal-floating-point: an unquoted string of characters from the set [0..9] and '.' which expresses a floating-point number in decimal positional notation.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1083
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- quoted-string: a string of characters within a pair of double-quotes ("), including Uniform Type Identifiers [UTI]. The set of characters allowed in the string and any rules for escaping special characters are specified by the Attribute definition, but any double-quote (") character and any carriage-return or linefeed will always be replaced by an escape sequence.

- enumerated-string: an unquoted character string from a set which is explicitly defined by the Attribute. An enumerated-string will never contain double-quotes ("), commas (,), or whitespace.

- decimal-resolution: two decimal-integers separated by the "x" character. The first integer is a horizontal pixel dimension (width); the second is a vertical pixel dimension (height).

The type of the AttributeValue for a given AttributeName is specified by the Attribute definition.

A given AttributeName MUST NOT appear more than once in a given Attribute List.

An Attribute/value pair with an unrecognized AttributeName MUST be ignored by the client.

Any tag containing an attribute/value pair of type enumerated-string whose AttributeName is recognized but whose AttributeValue is not recognized MUST be ignored by the client.

3.3. Standard Tags

3.3.1. EXTM3U

An Extended M3U file is distinguished from a basic M3U file by its first line, which MUST be the tag #EXTM3U.

3.3.2. EXTINF

The EXTINF tag specifies the duration of a media segment. It applies only to the media segment that follows it. Each media segment MUST be preceded by an EXTINF tag. Its format is:

#EXTINF:<duration>,<title>

"duration" is an integer or floating-point number in decimal positional notation that specifies the duration of the media segment in seconds. Durations that are reported as integers SHOULD be rounded to the nearest integer. Durations MUST be integers if the protocol version of the Playlist file is less than 3. The remainder

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1084
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

of the line following the comma is an optional human-readable informative title of the media segment.

3.4. New Tags

This document defines the following new tags: EXT-X-BYTERANGE, EXT-X-TARGETDURATION, EXT-X-MEDIA-SEQUENCE, EXT-X-KEY, EXT-X-PROGRAM-DATE-TIME, EXT-X-ALLOW-CACHE, EXT-X-PLAYLIST-TYPE, EXT-X-STREAM-INF, EXT-X-I-FRAME-STREAM-INF, EXT-X-I-FRAMES-ONLY, EXT-X-MEDIA, EXT-X-ENDLIST, EXT-X-DISCONTINUITY, and EXT-X-VERSION.

3.4.1. EXT-X-BYTERANGE

The EXT-X-BYTERANGE tag indicates that a media segment is a sub-range of the resource identified by its media URI. It applies only to the next media URI that follows it in the Playlist. Its format is:

#EXT-X-BYTERANGE:<n>[@o]

where n is a decimal-integer indicating the length of the sub-range in bytes. If present, o is a decimal-integer indicating the start of the sub-range, as a byte offset from the beginning of the resource. If o is not present, the sub-range begins at the next byte following the sub-range of the previous media segment.

If o is not present, a previous media segment MUST appear in the Playlist file and MUST be a sub-range of the same media resource.

A media URI with no EXT-X-BYTERANGE tag applied to it specifies a media segment that consists of the entire resource.

The EXT-X-BYTERANGE tag appeared in version 4 of the protocol.

3.4.2. EXT-X-TARGETDURATION

The EXT-X-TARGETDURATION tag specifies the maximum media segment duration. The EXTINF duration of each media segment in the Playlist file MUST be less than or equal to the target duration. This tag MUST appear once in the Playlist file. It applies to the entire Playlist file. Its format is:

#EXT-X-TARGETDURATION:<s>

where s is an integer indicating the target duration in seconds.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1085
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3.4.3. EXT-X-MEDIA-SEQUENCE

Each media segment in a Playlist has a unique integer sequence number. The sequence number of a segment is equal to the sequence number of the segment that preceded it plus one. The EXT-X-MEDIA-SEQUENCE tag indicates the sequence number of the first segment that appears in a Playlist file. Its format is:

#EXT-X-MEDIA-SEQUENCE:<number>

A Playlist file MUST NOT contain more than one EXT-X-MEDIA-SEQUENCE tag. If the Playlist file does not contain an EXT-X-MEDIA-SEQUENCE tag then the sequence number of the first segment in the playlist SHALL be considered to be 0. A client MUST NOT assume that segments with the same sequence number in different variants or renditions contain matching content.

A media URI is not required to contain its sequence number.

See Section 6.3.2 and Section 6.3.5 for information on handling the EXT-X-MEDIA-SEQUENCE tag.

3.4.4. EXT-X-KEY

Media segments MAY be encrypted. The EXT-X-KEY tag specifies how to decrypt them. It applies to every media segment that appears between it and the next EXT-X-KEY tag in the Playlist file with the same KEYFORMAT attribute (or the end of the Playlist file). Two or more EXT-X-KEY tags with different KEYFORMAT attributes MAY apply to the same media segment, in which case they MUST resolve to the same key. Its format is:

#EXT-X-KEY:<attribute-list>

The following attributes are defined:

METHOD

The value is an enumerated-string that specifies the encryption method. This attribute is mandatory.

The methods defined are: NONE, AES-128, and SAMPLE-AES.

An encryption method of NONE means that media segments are not encrypted. If the encryption method is NONE, the following attributes MUST NOT be present: URI; IV; KEYFORMAT; KEYFORMATVERSIONS.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1086
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

An encryption method of AES-128 means that media segments are completely encrypted using the Advanced Encryption Standard [AES_128] with a 128-bit key and PKCS7 padding [RFC5652]. If the encryption method is AES-128, the URI attribute MUST be present. The IV attribute MAY be present; see Section 5.2.

An encryption method of SAMPLE-AES means that the media segments contain elementary streams of audio, video, or other samples that are encrypted using the Advanced Encryption Standard [AES_128]. How an elementary stream is encrypted depends on the media encoding. The encryption format for H.264 [H_264], AAC [ISO_14496] and AC-3 [AC_3] elementary streams is described by [SampleEnc]. The IV attribute MAY be present; see Section 5.2.

URI

The value is a quoted-string containing a URI [RFC3986] that specifies how to obtain the key. This attribute is mandatory unless the METHOD is NONE.

IV

The value is a hexadecimal-integer that specifies the Initialization Vector to be used with the key. The IV attribute appeared in protocol version 2.

KEYFORMAT

The value is a quoted-string that specifies how the key is represented in the resource identified by the URI; see Section 5 for more detail. This attribute is optional; its absence indicates, an implicit value of "identity". The KEYFORMAT attribute appeared in protocol version 5.

KEYFORMATVERSIONS

The value is a quoted-string containing one or more positive integers separated by the "/" character (for example, "1/3"). If more than one version of a particular KEYFORMAT is defined, this attribute can be used to indicate which version(s) this instance complies with. This attribute is optional; if it is not present, its value is considered to be "1". The KEYFORMATVERSIONS attribute appeared in protocol version 5.

If the Playlist file does not contain an EXT-X-KEY tag then media segments are not encrypted.

See Section 5 for the format of the key file, and Section 5.2,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1087
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Section 6.2.3 and Section 6.3.6 for additional information on media segment encryption.

3.4.5. EXT-X-PROGRAM-DATE-TIME

The EXT-X-PROGRAM-DATE-TIME tag associates the first sample of a media segment with an absolute date and/or time. It applies only to the next media segment.

The date/time representation is ISO/IEC 8601:2004 [ISO_8601] and SHOULD indicate a time zone:

#EXT-X-PROGRAM-DATE-TIME:<YYYY-MM-DDThh:mm:ssZ>

For example:

#EXT-X-PROGRAM-DATE-TIME:2010-02-19T14:54:23.031+08:00

See Section 6.2.1 and Section 6.3.3 for more information on the EXT-X-PROGRAM-DATE-TIME tag.

3.4.6. EXT-X-ALLOW-CACHE

The EXT-X-ALLOW-CACHE tag indicates whether the client MAY or MUST NOT cache downloaded media segments for later replay. It MAY occur anywhere in the Playlist file; it MUST NOT occur more than once. The EXT-X-ALLOW-CACHE tag applies to all segments in the playlist. Its format is:

#EXT-X-ALLOW-CACHE:<YES|NO>

See Section 6.3.3 for more information on the EXT-X-ALLOW-CACHE tag.

3.4.7. EXT-X-PLAYLIST-TYPE

The EXT-X-PLAYLIST-TYPE tag provides mutability information about the Playlist file. It applies to the entire Playlist file. It is optional. Its format is:

#EXT-X-PLAYLIST-TYPE:<EVENT|VOD>

Section 6.2.1 defines the implications of the EXT-X-PLAYLIST-TYPE tag.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1088
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3.4.8. EXT-X-ENDLIST

The EXT-X-ENDLIST tag indicates that no more media segments will be added to the Playlist file. It MAY occur anywhere in the Playlist file; it MUST NOT occur more than once. Its format is:

#EXT-X-ENDLIST

3.4.9. EXT-X-MEDIA

The EXT-X-MEDIA tag is used to relate Playlists that contain alternative renditions of the same content. For example, three EXT-X-MEDIA tags can be used to identify audio-only Playlists that contain English, French and Spanish renditions of the same presentation. Or two EXT-X-MEDIA tags can be used to identify video-only Playlists that show two different camera angles.

The EXT-X-MEDIA tag stands alone, in that it does not apply to a particular URI in the Playlist. Its format is:

#EXT-X-MEDIA:<attribute-list>

The following attributes are defined:

URI

The value is a quoted-string containing a URI that identifies the Playlist file. This attribute is optional; see Section 3.4.10.1.

TYPE

The value is enumerated-string; valid strings are AUDIO, VIDEO and SUBTITLES. If the value is AUDIO, the Playlist described by the tag MUST contain audio media. If the value is VIDEO, the Playlist MUST contain video media. If the value is SUBTITLES, the Playlist MUST contain subtitle media.

GROUP-ID

The value is a quoted-string identifying a mutually-exclusive group of renditions. The presence of this attribute signals membership in the group. See Section 3.4.9.1.

LANGUAGE

The value is a quoted-string containing an RFC 5646 [RFC5646] language tag that identifies the primary language used in the rendition. This attribute is optional.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1089
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

NAME

The value is a quoted-string containing a human-readable description of the rendition. If the LANGUAGE attribute is present then this description SHOULD be in that language.

DEFAULT

The value is an enumerated-string; valid strings are YES and NO. If the value is YES, then the client SHOULD play this rendition of the content in the absence of information from the user indicating a different choice. This attribute is optional. Its absence indicates an implicit value of NO.

AUTOSELECT

The value is an enumerated-string; valid strings are YES and NO. This attribute is optional. Its absence indicates an implicit value of NO. If the value is YES, then the client MAY choose to play this rendition in the absence of explicit user preference because it matches the current playback environment, such as chosen system language.

FORCED

The value is an enumerated-string; valid strings are YES and NO. This attribute is optional. Its absence indicates an implicit value of NO. The FORCED attribute MUST NOT be present unless the TYPE is SUBTITLES.

A value of YES indicates that the rendition contains content which is considered essential to play. When selecting a FORCED rendition, a client should choose the one that best matches the current playback environment (e.g. language).

A value of NO indicates that the rendition contains content which is intended to be played in response to explicit user request.

CHARACTERISTICS

The value is a quoted-string containing one or more Uniform Type Identifiers [UTI] separated by comma (,) characters. This attribute is optional. Each UTI indicates an individual characteristic of the rendition.

A SUBTITLES rendition MAY include the following characteristics: "public.accessibility.transcribes-spoken-dialog"; "public.accessibility.describes-music-and-sound"; "public.easy-to-

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1090
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

read" (which indicates that the subtitles have been edited for ease of reading).

An AUDIO rendition MAY include the following characteristics: "public.accessibility.describes-video".

The CHARACTERISTICS attribute MAY include private UTIs.

The EXT-X-MEDIA tag appeared in version 4 of the protocol.

3.4.9.1. Rendition Groups

A set of EXT-X-MEDIA tags with the same GROUP-ID value forms a group of renditions. Each member of the group MUST represent an alternative rendition of the same content.

All EXT-X-MEDIA tags in a Playlist MUST meet the following constraints:

- All EXT-X-MEDIA tags in the same group MUST have the same TYPE attribute.
- All EXT-X-MEDIA tags in the same group MUST have different NAME attributes.
- A group MUST NOT have more than one member with a DEFAULT attribute of YES.
- All members of a group whose AUTOSELECT attribute has a value of YES MUST have LANGUAGE [RFC5646] attributes with unique values.
- All members of a group with TYPE=AUDIO MUST use the same audio sample format.
- All members of a group with TYPE=VIDEO MUST use the same video sample format.

A Playlist MAY contain multiple groups of the same TYPE in order to provide multiple encodings of each rendition. If it does so, each group of the same TYPE SHOULD contain corresponding members with the same NAME attribute, LANGUAGE attribute, and rendition.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1091
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3.4.10. EXT-X-STREAM-INF

The EXT-X-STREAM-INF tag identifies a media URI as a Playlist file containing a multimedia presentation and provides information about that presentation. It applies only to the URI that follows it. Its format is:

```
#EXT-X-STREAM-INF:<attribute-list>
<URI>
```

The following attributes are defined:

BANDWIDTH

The value is a decimal-integer of bits per second. It MUST be an upper bound of the overall bitrate of each media segment (calculated to include container overhead) that appears or will appear in the Playlist.

Every EXT-X-STREAM-INF tag MUST include the BANDWIDTH attribute.

PROGRAM-ID

The value is a decimal-integer that uniquely identifies a particular presentation within the scope of the Playlist file.

A Playlist file MAY contain multiple EXT-X-STREAM-INF tags with the same PROGRAM-ID to identify different encodings of the same presentation. These variant playlists MAY contain additional EXT-X-STREAM-INF tags.

CODECS

The value is a quoted-string containing a comma-separated list of formats, where each format specifies a media sample type that is present in a media segment in the Playlist file. Valid format identifiers are those in the ISO File Format Name Space defined by RFC 6381 [RFC6381].

Every EXT-X-STREAM-INF tag SHOULD include a CODECS attribute.

RESOLUTION

The value is a decimal-resolution describing the approximate encoded horizontal and vertical resolution of video within the presentation.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1092
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

AUDIO

The value is a quoted-string. It MUST match the value of the GROUP-ID attribute of an EXT-X-MEDIA tag elsewhere in the Playlist whose TYPE attribute is AUDIO. It indicates the set of audio renditions that MAY be used when playing the presentation. See Section 3.4.10.1.

VIDEO

The value is a quoted-string. It MUST match the value of the GROUP-ID attribute of an EXT-X-MEDIA tag elsewhere in the Playlist whose TYPE attribute is VIDEO. It indicates the set of video renditions that MAY be used when playing the presentation. See Section 3.4.10.1.

SUBTITLES

The value is a quoted-string. It MUST match the value of the GROUP-ID attribute of an EXT-X-MEDIA tag elsewhere in the Playlist whose TYPE attribute is SUBTITLES. It indicates the set of subtitle renditions that MAY be used when playing the presentation. See Section 3.4.10.1.

3.4.10.1. Alternative Renditions

When an EXT-X-STREAM-INF tag contains an AUDIO, VIDEO, or SUBTITLE attribute, it indicates that alternative renditions of the content are available for playback of that variant.

When defining alternative renditions, the following constraints MUST be met:

- All playable combinations of renditions associated with an EXT-X-STREAM-INF tag MUST have an aggregate bandwidth less than or equal to the BANDWIDTH attribute of the EXT-X-STREAM-INF tag.

- If an EXT-X-STREAM-INF tag contains a RESOLUTION attribute and a VIDEO attribute, then every alternative video rendition MUST match the value of the RESOLUTION attribute.

- Every alternative rendition associated with an EXT-X-STREAM-INF tag MUST meet the constraints for a variant stream described in Section 6.2.4.

The URI attribute of an EXT-X-MEDIA tag is optional. If it is missing, it indicates that the rendition described by the EXT-X-MEDIA tag is present in the main Playlist described by the EXT-X-STREAM-INF

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1093
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

tag.

Note that if a client chooses to play renditions of audio and video that are not present in the main Playlist described by the EXT-X-STREAM-INF tag, or if the client chooses to play an audio rendition and the main Playlist is audio-only, then the client MAY ignore the main Playlist and its media.

3.4.11. EXT-X-DISCONTINUITY

The EXT-X-DISCONTINUITY tag indicates an encoding discontinuity between the media segment that follows it and the one that preceded it. The set of characteristics that MAY change is:

- file format
- number and type of tracks
- encoding parameters
- encoding sequence
- timestamp sequence

Its format is:

#EXT-X-DISCONTINUITY

See Section 4, Section 6.2.1, and Section 6.3.3 for more information about the EXT-X-DISCONTINUITY tag.

3.4.12. EXT-X-I-FRAMES-ONLY

The EXT-X-I-FRAMES-ONLY tag indicates that each media segment in the Playlist describes a single I-frame. I-frames (or Intra frames) are encoded video frames whose encoding does not depend on any other frame.

The EXT-X-I-FRAMES-ONLY tag applies to the entire Playlist. Its format is:

#EXT-X-I-FRAMES-ONLY

In a Playlist with the EXT-X-I-FRAMES-ONLY tag, the media segment duration (EXTINF tag value) is the time between the presentation time of the I-frame in the media segment and the presentation time of the next I-frame in the Playlist, or the end of the presentation if it is the last I-frame in the Playlist.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1094
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Media resources containing I-frame segments MUST begin with either a Transport Stream PAT/PMT or be accompanied by an EXT-X-MAP tag indicating the proper PAT/PMT. The byte range of an I-frame segment with an EXT-X-BYTERANGE tag applied to it (Section 3.4.1) MUST NOT include a PAT/PMT.

The EXT-X-I-FRAMES-ONLY tag appeared in version 4 of the protocol.

3.4.13. EXT-X-MAP

The EXT-X-MAP tag specifies how to obtain the Transport Stream PAT/PMT for the applicable media segment. It applies to every media segment that appears after it in the Playlist until the next EXT-X-DISCONTINUITY tag, or until the end of the playlist.

The EXT-X-MAP tag MUST NOT appear unless the Playlist also contains the EXT-X-I-FRAMES-ONLY tag. It is RECOMMENDED that the EXT-X-MAP tag only be used for segments whose resource does not start with a PAT/PMT.

Its format is:

#EXT-X-MAP:<attribute-list>

The following attributes are defined:

URI

The value is a quoted-string containing a URI that identifies a resource that contains the Transport Stream PAT/PMT. This attribute is mandatory.

BYTERANGE

The value is a quoted-string specifying a byte range into the resource identified by the URI attribute. This range SHOULD contain only the Transport Stream PAT/PMT. The format of the byte range is described in Section 3.4.1. This attribute is optional; if it is not present, the byte range is the entire resource indicated by the URI.

The EXT-X-MAP tag appeared in version 5 of the protocol.

3.4.14. EXT-X-I-FRAME-STREAM-INF

The EXT-X-I-FRAME-STREAM-INF tag identifies a Playlist file containing the I-frames of a multimedia presentation. It stands alone, in that it does not apply to a particular URI in the Playlist. Its format is:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1095
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

#EXT-X-I-FRAME-STREAM-INF:<attribute-list>

All attributes defined for the EXT-X-STREAM-INF tag (Section 3.4.10) are also defined for the EXT-X-I-FRAME-STREAM-INF tag, except for the AUDIO and SUBTITLES attributes. In addition, the following attribute is defined:

URI

The value is a quoted-string containing a URI that identifies the I-frame Playlist file.

Every EXT-X-I-FRAME-STREAM-INF tag MUST include a BANDWIDTH attribute and a URI attribute.

The provisions in Section 3.4.10.1 also apply to EXT-X-I-FRAME-STREAM-INF tags with a VIDEO attribute.

A Playlist that specifies alternative VIDEO renditions and I-frame Playlists SHOULD include an alternative I-frame VIDEO rendition for each regular VIDEO rendition, with the same NAME and LANGUAGE attributes.

The EXT-X-I-FRAME-STREAM-INF tag appeared in version 4 of the protocol. Clients that do not implement protocol version 4 or higher MUST ignore it.

3.4.15. EXT-X-VERSION

The EXT-X-VERSION tag indicates the compatibility version of the Playlist file. The Playlist file, its associated media, and its server MUST comply with all provisions of the most-recent version of this document describing the protocol version indicated by the tag value.

The EXT-X-VERSION tag applies to the entire Playlist file. Its format is:

#EXT-X-VERSION:<n>

where n is an integer indicating the protocol version.

A Playlist file MUST NOT contain more than one EXT-X-VERSION tag. A Playlist file that does not contain an EXT-X-VERSION tag MUST comply with version 1 of this protocol.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1096
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

4. Media segments

Each media URI in a Playlist file specifies a media segment which is part of the overall presentation. If a media URI has an EXT-X-BYTERANGE tag applied to it, the segment is a sub-range of the media file identified by the URI. Otherwise, the segment is the entire media file.

Each media segment MUST be formatted as an MPEG-2 Transport Stream [ISO_13818], an MPEG audio elementary stream [ISO_11172], or a WebVTT [WebVTT] file.

Transport Stream segments MUST contain a single MPEG-2 Program. There SHOULD be a Program Association Table (PAT) and a Program Map Table (PMT) at the start of each segment. A segment that contains video SHOULD have at least one key frame and enough information to completely initialize a video decoder.

A Transport Stream or audio elementary stream segment MUST be the continuation of the encoded media at the end of the segment with the previous sequence number, where values in a continuous series, such as timestamps and Continuity Counters, continue uninterrupted - unless the media segment was the first ever to appear in the Playlist file or has an EXT-X-DISCONTINUITY tag applied to it.

Clients SHOULD be prepared to handle multiple tracks of a particular type (e.g. audio or video). A client with no other preference SHOULD choose the track with the lowest numerical PID that it can play.

Clients MUST ignore private streams inside Transport Streams that they do not recognize.

The encoding parameters for samples in a stream inside a media segment and between corresponding streams across multiple media segments SHOULD remain consistent. However clients SHOULD deal with encoding changes as they are encountered, for example by scaling video content to accommodate a resolution change.

Subtitle segments MUST be formatted as WebVTT [WebVTT] files. Each subtitle segment MUST contain all subtitle cues that are intended to be displayed during the period indicated by the segment EXTINF duration. The start time offset and end time offset of each cue MUST indicate the total display time for that cue, even if that time range extends beyond the EXTINF duration. A WebVTT segment MAY contain no cues; this indicates that no subtitles are to be displayed during that period.

Each WebVTT segment MUST have an X-TIMESTAMP-MAP metadata header.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1097
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

This header synchronizes the cue timestamps in the WebVTT file with the MPEG-2 (PES) timestamps in other streams. Its format is:

X-TIMESTAMP-MAP=LOCAL:<cue time>,MPEGTS:<MPEG-2 time>
e.g. X-TIMESTAMP-MAP=LOCAL:00:00:00.000,MPEGTS:900000

The cue timestamp in the LOCAL attribute MAY fall outside the range of time covered by the segment.

5. Key files

5.1. Introduction

An EXT-X-KEY tag with a URI attribute identifies a Key file. A Key file contains the cipher key that MUST be used to decrypt subsequent media segments in the Playlist.

[AES_128] encryption uses 16-octet keys. If the KEYFORMAT of an EXT-X-KEY tag is "identity", the Key file is a single packed array of 16 octets in binary format.

5.2. IV for [AES_128]

[AES_128] requires the same 16-octet Initialization Vector (IV) to be supplied when encrypting and decrypting. Varying this IV increases the strength of the cipher.

If an EXT-X-KEY tag has a KEYFORMAT of "identity" and an IV attribute is present, implementations MUST use the attribute value as the IV when encrypting or decrypting with that key. The value MUST be interpreted as a 128-bit number.

If an EXT-X-KEY tag with a KEYFORMAT of "identity" does not have the IV attribute, implementations MUST use the sequence number of the media segment as the IV when encrypting or decrypting that media segment. The big-endian binary representation of the sequence number SHALL be placed in a 16-octet buffer and padded (on the left) with zeros.

6. Client/Server Actions

6.1. Introduction

This section describes how the server generates the Playlist and media segments and how the client should download and play them.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1098
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

6.2. Server Process

6.2.1. Introduction

The production of the media stream is outside the scope of this document, which simply presumes a source of a continuous stream containing the presentation.

The server MUST divide the stream into individual media segments whose duration is less than or equal to a constant target duration. The server SHOULD attempt to divide the stream at points that support effective decode of individual media segments, e.g. on packet and key frame boundaries.

The server MUST create a URI for every media segment that enables its clients to obtain the segment data. If a server supports partial loading of resources (e.g. via HTTP Range requests), it MAY specify segments as sub-ranges of larger resources using the EXT-X-BYTERANGE tag.

If WebVTT segments are distributed by HTTP, the server SHOULD support client requests to use the "gzip" Content-Encoding.

The server MUST create a Playlist file. The Playlist file MUST conform to the format described in Section 3. A URI for each media segment that the server wishes to make available MUST appear in the Playlist in the order in which it is to be played. The entire media segment MUST be available to clients if its URI is in the Playlist file.

The Playlist file MUST contain an EXT-X-TARGETDURATION tag. Its value MUST be equal to or greater than the EXTINF value of any media segment that appears or will appear in the Playlist file. Its value MUST NOT change. A typical target duration is 10 seconds.

The Playlist file SHOULD contain one EXT-X-VERSION tag which indicates the compatibility version of the stream. Its value MUST be the lowest protocol version with which the server, Playlist file, and associated media segments all comply. Its value MUST NOT change.

The server MUST create a URI for the Playlist file that will allow its clients to obtain the file.

If the Playlist file is distributed by HTTP, the server SHOULD support client requests to use "gzip" Content-Encoding.

Changes to the Playlist file MUST be made atomically from the point of view of the clients.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1099
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The server MUST NOT change the Playlist file, except to:

Append lines to it (Section 6.2.1).

Remove media segment URIs from the Playlist in the order that they appear, along with any tags that apply only to those segments (Section 6.2.2).

Increment the value of the EXT-X-MEDIA-SEQUENCE tag (Section 6.2.2).

Add or remove EXT-X-STREAM-INF tags or EXT-X-I-FRAME-STREAM-INF tags (Section 6.2.4). Note that clients are not required to reload variant Playlist files, so changing them may not have immediate effect.

Add an EXT-X-ENDLIST tag to the Playlist (Section 6.2.1).

Furthermore, the Playlist file MAY contain an EXT-X-PLAYLIST-TYPE tag with a value of either EVENT or VOD. If the tag is present and has a value of EVENT, the server MUST NOT change or delete any part of the Playlist file (although it MAY append lines to it). If the tag is present and has a value of VOD, the Playlist file MUST NOT change.

Every media segment in a Playlist MUST have an EXTINF tag applied to it indicating the duration of the media segment.

The server MAY associate an absolute date and time with a media segment by applying an EXT-X-PROGRAM-DATE-TIME tag to the segment. The date and time value provides an informative mapping of the timeline of the media to an appropriate wall-clock time, which may be used as a basis for seeking, for display, or for other purposes. If a server provides this mapping, it SHOULD apply an EXT-X-PROGRAM-DATE-TIME tag to every segment that has an EXT-X-DISCONTINUITY tag applied to it.

If the Playlist contains the final media segment of the presentation then the Playlist file MUST contain the EXT-X-ENDLIST tag.

If the Playlist does not contain the EXT-X-ENDLIST tag, the server MUST make a new version of the Playlist file available that contains at least one new media segment. It MUST be made available relative to the time that the previous version of the Playlist file was made available: no earlier than one-half the target duration after that time, and no later than 1.5 times the target duration after that time.

If the server wishes to remove an entire presentation, it MUST make

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1100
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

the Playlist file unavailable to clients. It SHOULD ensure that all media segments in the Playlist file remain available to clients for at least the duration of the Playlist file at the time of removal.

6.2.2. Sliding Window Playlists

The server MAY limit the availability of media segments by removing media segments from the Playlist file (Section 6.2.1). If media segments are to be removed, the Playlist file MUST contain exactly one EXT-X-MEDIA-SEQUENCE tag. Its value MUST be incremented by 1 for every media segment that is removed from the Playlist file.

Media segments MUST be removed from the Playlist file in the order that they appear in the Playlist.

The server MUST NOT remove a media segment from the Playlist file if the duration of the Playlist file minus the duration of the segment is less than three times the target duration.

When the server removes a media segment from the Playlist, the corresponding media URI SHOULD remain available to clients for a period of time equal to the duration of the segment plus the duration of the longest Playlist file distributed by the server containing that segment.

If a server plans to remove a media segment after it is delivered to clients over HTTP, it SHOULD ensure that the HTTP response contains an Expires header that reflects the planned time-to-live.

6.2.3. Encrypting media segments

If media segments are to be encrypted the server MUST define a URI which will allow authorized clients to obtain a Key file containing a decryption key. The Key file MUST conform to the format described in Section 5.

The server MAY set the HTTP Expires header in the key response to indicate that the key may be cached.

The server MUST encrypt every media segment in a Playlist according to the EXT-X-KEY tag that applies to its URI in the Playlist file. Media segments with an EXT-X-KEY tag whose METHOD is NONE, or which do not have an EXT-X-KEY tag applied to them, MUST NOT be encrypted.

If the encryption METHOD is AES-128 and the Playlist does not contain the EXT-X-I-FRAMES-ONLY tag, AES-128 CBC encryption with PKCS7 padding [RFC5652] SHALL be applied to individual media segments. The entire segment MUST be encrypted. Cipher Block Chaining MUST NOT be

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1101
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

applied across media segments. The IV used for encryption MUST be either the sequence number of the media segment or the value of the IV attribute of the EXT-X-KEY tag, as described in Section 5.2.

If the encryption METHOD is AES-128 and the Playlist contains an EXT-X-I-FRAMES-ONLY tag, AES-128 CBC encryption with PKCS7 padding [RFC5652] MUST be applied to the entire resource. The entire resource MUST be encrypted. Encryption MAY be restarted on 16-byte block boundaries, unless the first block contains an I-frame. The IV used for encryption MUST be either the sequence number of the media segment or the value of the IV attribute of the EXT-X-KEY tag, as described in Section 5.2.

If the encryption METHOD is SAMPLE-AES, certain elementary streams MAY be encrypted prior to encapsulation in a media segment. The encryption format for H.264, AAC and AC-3 elementary streams is described by [SampleEnc].

The server MUST NOT remove an EXT-X-KEY tag from the Playlist file if it applies to any media segment in the Playlist file.

6.2.4. Providing variant streams

A server MAY offer multiple Playlist files to provide different encodings of the same presentation. If it does so it SHOULD provide a variant Playlist file that lists each variant stream to allow clients to switch between encodings dynamically.

Variant Playlists MUST contain an EXT-X-STREAM-INF tag or EXT-X-I-FRAME-STREAM-INF tag for each variant stream. Each tag identifying an encoding of the same presentation MUST have the same PROGRAM-ID attribute value. The PROGRAM-ID value for each presentation MUST be unique within the variant Playlist.

If an EXT-X-STREAM-INF tag or EXT-X-I-FRAME-STREAM-INF tag contains the CODECS attribute, the attribute value MUST include every format defined by [RFC6381] that is present in any media segment that appears or will appear in the Playlist file.

The server MUST meet the following constraints when producing variant streams:

- Each variant stream MUST present the same content, including stream discontinuities.

- Each variant Playlist file MUST have the same target duration. The only exception is that SUBTITLES renditions with a EXT-X-PLAYLIST-TYPE of VOD MAY have longer target durations.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1102
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Content that appears in one variant Playlist file but not in another MUST appear either at the beginning or at the end of the Playlist file and MUST NOT be longer than the target duration.

Matching content in variant streams MUST have matching timestamps. This allows clients to synchronize the streams.

Each Elementary Audio Stream segment MUST signal the timestamp of its first sample with an ID3 PRIV tag [ID3] at the beginning of the segment. The ID3 PRIV owner identifier MUST be "com.apple.streaming.transportStreamTimestamp". The ID3 payload MUST be a 33-bit MPEG-2 Program Elementary Stream timestamp expressed as a big-endian eight-octet number, with the upper 31 bits set to zero.

In addition, all variant streams SHOULD contain the same encoded audio bitstream. This allows clients to switch between streams without audible glitching.

The rules for variant streams also apply to alternate renditions - see Section 3.4.10.1.

6.3. Client Process

6.3.1. Introduction

How the client obtains the URI to the Playlist file is outside the scope of this document; it is presumed to have done so.

The client MUST obtain the Playlist file from the URI. If the Playlist file so obtained is a variant Playlist, the client MUST obtain the Playlist file from the variant Playlist.

This document does not specify the treatment of variant streams by clients.

6.3.2. Loading the Playlist file

Every time a Playlist file is loaded or reloaded from the Playlist URI:

The client MUST ensure that the Playlist file begins with the EXTM3U tag and that the EXT-X-VERSION tag, if present, specifies a protocol version supported by the client; if not, the client MUST NOT attempt to use the Playlist.

The client SHOULD ignore any tags and attributes it does not recognize.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1103
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The client MUST determine the next media segment to load, as described in Section 6.3.5.

If the Playlist contains the EXT-X-MEDIA-SEQUENCE tag, the client SHOULD assume that each media segment in it will become unavailable at the time that the Playlist file was loaded plus the duration of the Playlist file.

6.3.3. Playing the Playlist file

The client SHALL choose which media segment to play first from the Playlist when playback starts. If the EXT-X-ENDLIST tag is not present and the client intends to play the media regularly (i.e. in playlist order at the nominal playback rate), the client SHOULD NOT choose a segment which starts less than three target durations from the end of the Playlist file. Doing so can trigger playback stalls.

To achieve regular playback, media segments MUST be played in the order that they appear in the Playlist file. The client MAY present the available media in any way it wishes, including regular playback, random access, and trick modes.

The client MUST be prepared to reset its parser(s) and decoder(s) before playing a media segment that has an EXT-X-DISCONTINUITY tag applied to it.

The client SHOULD attempt to load media segments in advance of when they will be required for uninterrupted playback to compensate for temporary variations in latency and throughput.

If the Playlist file contains the EXT-X-ALLOW-CACHE tag and its value is NO, the client MUST NOT cache downloaded media segments after they have been played. Otherwise the client MAY cache downloaded media segments indefinitely for later replay.

The client MAY use the value of the EXT-X-PROGRAM-DATE-TIME tag to display the program origination time to the user. If the value includes time zone information the client SHALL take it into account, but if it does not the client MUST NOT infer an originating time zone.

The client MUST NOT depend upon the correctness or the consistency of the value of the EXT-X-PROGRAM-DATE-TIME tag.

6.3.4. Reloading the Playlist file

The client MUST periodically reload the Playlist file unless it contains the EXT-X-ENDLIST tag.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1104
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

However the client MUST NOT attempt to reload the Playlist file more frequently than specified by this section.

When a client loads a Playlist file for the first time or reloads a Playlist file and finds that it has changed since the last time it was loaded, the client MUST wait for a period of time before attempting to reload the Playlist file again. This period is called the initial minimum reload delay. It is measured from the time that the client began loading the Playlist file.

The initial minimum reload delay is the duration of the last media segment in the Playlist. Media segment duration is specified by the EXTINF tag.

If the client reloads a Playlist file and finds that it has not changed then it MUST wait for a period of one-half the target duration before retrying.

In order to reduce server load, the client SHOULD NOT reload the Playlist files of variant streams that are not currently being played. If it decides to switch playback to a different variant, it SHOULD stop reloading the Playlist of the old variant and begin loading the Playlist of the new variant. It can use the EXTINF durations and the constraints in Section 6.2.4 to determine the approximate location of corresponding media. Once media from the new variant has been loaded, the timestamps in the media segments can be used to synchronize the old and new timelines precisely. A client MUST NOT assume that segments with the same media sequence number in different variants or renditions contain matching content.

6.3.5. Determining the next segment to load

The client MUST examine the Playlist file every time it is loaded or reloaded to determine the next media segment to load.

The first segment to load MUST be the segment that the client has chosen to play first, as described in Section 6.3.3.

If the first segment to be played has been loaded and the Playlist file does not contain the EXT-X-MEDIA-SEQUENCE tag then the client MUST verify that the current Playlist file contains the URI of the last loaded media segment at the offset it was originally found at, halting playback if it does not. The next media segment to load MUST be the first media segment following the last-loaded segment in the Playlist.

If the first segment to be played has been loaded and the Playlist file contains the EXT-X-MEDIA-SEQUENCE tag then the next media

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1105
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

segment to load SHALL be the one with the lowest sequence number that is greater than the sequence number of the last media segment loaded.

6.3.6. Decrypting encrypted media segments

If a Playlist file contains an EXT-X-KEY tag that specifies a Key file URI, the client MUST obtain that key file and use the key inside it to decrypt all media segments to which that EXT-X-KEY tag applies.

A client MUST NOT attempt to use an EXT-X-KEY tag with an unsupported or unrecognized KEYFORMAT attribute. A client SHOULD fail playback if the Playlist contains a media segment to which only EXT-X-KEY tags with unrecognized or unsupported KEYFORMAT attributes are applied.

If the encryption METHOD is AES-128, AES-128 CBC decryption SHALL be applied to individual media segments. The entire segment MUST be decrypted. Cipher Block Chaining MUST NOT be applied across media segments. The IV used for decryption MUST be either the sequence number of the media segment or the value of the IV attribute of the EXT-X-KEY tag, as described in Section 5.2.

If the encryption METHOD is AES-128 and the media segment is part of an I-frame playlist (Section 3.4.12) special care MUST be taken in loading and decrypting the segment, because the resource identified by the URI is encrypted in 16-byte blocks from the start of the resource (offset 0). The sub-range specified by the EXT-X-BYTERANGE tag MUST be widened to include the 16-byte blocks in which the beginning and end of the sub-range fall. Next, it MUST be widened further to include the previous 16-byte block. That range MUST be loaded and decrypted with AES-128 CBC using an arbitrary IV. The decrypted segment will then be in the original (unwidened) sub-range.

If the encryption METHOD is SAMPLE-AES, AES-128 decryption SHALL be applied to encrypted elementary streams within the media segment. The encryption format for H.264, AAC and AC-3 elementary streams is described by [SampleEnc].

An EXT-X-KEY tag with a METHOD of NONE indicates that the media segments it applies to are not encrypted.

7. Protocol version compatibility

Clients and servers MUST implement protocol version 2 or higher to use:

- The IV attribute of the EXT-X-KEY tag.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1106
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Clients and servers MUST implement protocol version 3 or higher to use:

- Floating-point EXTINF duration values.

Clients and servers MUST implement protocol version 4 or higher to use:

- The EXT-X-BYTERANGE tag.
- The EXT-X-I-FRAME-STREAM-INF tag.
- The EXT-X-I-FRAMES-ONLY tag.
- The EXT-X-MEDIA tag.
- The AUDIO and VIDEO attributes of the EXT-X-STREAM-INF tag.

Clients and servers MUST implement protocol version 5 or higher to use:

- The KEYFORMAT and KEYFORMATVERSIONS attributes of the EXT-X-KEY tag.
- The EXT-X-MAP tag.

8. Examples

8.1. Introduction

This section contains several example Playlist files.

8.2. Simple Playlist file

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:5220
#EXTINF:5219.2,
http://media.example.com/entire.ts
#EXT-X-ENDLIST
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1107
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8.3. Sliding Window Playlist, using HTTPS

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:8
#EXT-X-MEDIA-SEQUENCE:2680

#EXTINF:7.975,
https://priv.example.com/fileSequence2680.ts
#EXTINF:7.941,
https://priv.example.com/fileSequence2681.ts
#EXTINF:7.975,
https://priv.example.com/fileSequence2682.ts
```

8.4. Playlist file with encrypted media segments

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:7794
#EXT-X-TARGETDURATION:15

#EXT-X-KEY:METHOD=AES-128,URI="https://priv.example.com/key.php?r=52"

#EXTINF:2.833,
http://media.example.com/fileSequence52-A.ts
#EXTINF:15.0,
http://media.example.com/fileSequence52-B.ts
#EXTINF:13.333,
http://media.example.com/fileSequence52-C.ts

#EXT-X-KEY:METHOD=AES-128,URI="https://priv.example.com/key.php?r=53"

#EXTINF:15.0,
http://media.example.com/fileSequence53-A.ts
```

8.5. Variant Playlist file

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1280000
http://example.com/low.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2560000
http://example.com/mid.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
http://example.com/hi.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=65000,CODECS="mp4a.40.5"
http://example.com/audio-only.m3u8
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1108
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8.6. Variant Playlist with I-Frames

In this example, the PROGRAM-ID attributes have been left out:

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1280000
low/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=86000,URI="low/iframe.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=2560000
mid/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=150000,URI="mid/iframe.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=7680000
hi/audio-video.m3u8
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=550000,URI="hi/iframe.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5"
audio-only.m3u8
```

8.7. Variant Playlist with Alternative audio

In this example, the PROGRAM-ID attributes have been left out. The CODECS attributes have been condensed for space. A '\' is used to indicate that the tag continues on the following line with whitespace removed:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="English", \
   DEFAULT=YES,AUTOSELECT=YES,LANGUAGE="en", \
   URI="main/english-audio.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="Deutsche", \
   DEFAULT=NO,AUTOSELECT=YES,LANGUAGE="de", \
   URI="main/german-audio.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="Commentary", \
   DEFAULT=NO,AUTOSELECT=NO,URI="commentary/audio-only.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=1280000,CODECS="...",AUDIO="aac"
low/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2560000,CODECS="...",AUDIO="aac"
mid/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=7680000,CODECS="...",AUDIO="aac"
hi/video-only.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5",AUDIO="aac"
main/english-audio.m3u8
```

8.8. Variant Playlist with Alternative video

In this example, the PROGRAM-ID attributes have been left out. The CODECS attributes have been condensed for space. A '\' is used to indicate that the tag continues on the following line with whitespace removed:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1109
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
#EXTM3U
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="low",NAME="Main", \
   DEFAULT=YES,URI="low/main/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="low",NAME="Centerfield", \
   DEFAULT=NO,URI="low/centerfield/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="low",NAME="Dugout", \
   DEFAULT=NO,URI="low/dugout/audio-video.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=1280000,CODECS="...",VIDEO="low"
low/main/audio-video.m3u8

#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="mid",NAME="Main", \
   DEFAULT=YES,URI="mid/main/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="mid",NAME="Centerfield", \
   DEFAULT=NO,URI="mid/centerfield/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="mid",NAME="Dugout", \
   DEFAULT=NO,URI="mid/dugout/audio-video.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=2560000,CODECS="...",VIDEO="mid"
mid/main/audio-video.m3u8

#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="hi",NAME="Main", \
   DEFAULT=YES,URI="hi/main/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="hi",NAME="Centerfield", \
   DEFAULT=NO,URI="hi/centerfield/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="hi",NAME="Dugout", \
   DEFAULT=NO,URI="hi/dugout/audio-video.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=7680000,CODECS="...",VIDEO="hi"
hi/main/audio-video.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5"
main/audio-only.m3u8
```

9. Contributors

Significant contributions to the design of this protocol were made by Jim Batson, David Biderman, Bill May, Roger Pantos, and Alan Tseng.

10. IANA Considerations

This memo requests that the following MIME type [RFC2046] be registered with the IANA:

Type name: "application"

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1110
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Subtype name: "vnd.apple.mpegurl"

Required parameters: (none)

Optional parameters: (none)

Encoding considerations: encoded as text. See Section 3 for more information.

Security considerations: See Section 11.

Compression: this media type does not employ compression.

Interoperability considerations: There are no byte-ordering issues, since files are 7- or 8-bit text. Applications could encounter unrecognized tags, which SHOULD be ignored.

Published specification: see Section 3.

Applications that use this media type: Multimedia applications such as the iPhone media player in iOS 3.0 and later and QuickTime Player in Mac OS X version 10.6 and later.

Additional information: files begin with the magic number #EXTM3U. Filenames normally end with .m3u8 or .m3u (see Section 3). No Macintosh file type codes have been registered.

Person & email address to contact for further information: David Singer, singer AT apple.com.

Intended usage: LIMITED USE

Restrictions on usage: (none)

Author: Roger Pantos

Change Controller: David Singer

11. Security Considerations

Since the protocol generally uses HTTP to transfer data, most of the same security considerations apply. See section 15 of RFC 2616 [RFC2616].

Media file parsers are typically subject to "fuzzing" attacks. Clients SHOULD take care when parsing segments received from a server that non-compliant segments are rejected.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1111
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Playlist files contain URIs, which clients will use to make network requests of arbitrary entities. Clients SHOULD range-check responses to prevent buffer overflows. See also the Security Considerations section of RFC 3986 [RFC3986].

Clients SHOULD load resources identified by URI lazily to avoid contributing to denial-of-service attacks.

HTTP requests often include session state ("cookies"), which may contain private user data. Implementations MUST follow cookie restriction and expiry rules specified by RFC 6265 [RFC6265]. See also the Security Considerations section of RFC 6265, and RFC 2964 [RFC2964].

Encryption keys are specified by URI. The delivery of these keys SHOULD be secured by a mechanism such as HTTP over TLS [RFC5246] (formerly SSL) in conjunction with a secure realm or a session cookie.

12. References

12.1. Normative References

[AC_3] Advanced Television Systems Committee, "ATSC Standard: A/52:2010: Digital Audio Compression (AC-3) (E-AC-3) Standard", November 2010, <http://www.atsc.org/cms/standards/a_52-2010.pdf>.

[AES_128] U.S. Department of Commerce/National Institute of Standards and Technology, "Advanced Encryption Standard (AES), FIPS PUB 197", November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[H_264] International Telecommunications Union, "Advanced video coding for generic audiovisual services", January 2012, <http://www.itu.int/rec/T-REC-H.264>.

[ISO_11172] International Organization for Standardization, "ISO/IEC International Standard 11172-1; Coding of moving pictures and associated audio for digital storage media -- Part 1: Systems", 1993, <http://www.iso.org/iso/catalogue_detail?csnumber=19180>.

[ISO_13818] International Organization for Standardization, "ISO/IEC International Standard 13818; Generic coding of moving

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1112
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

pictures and associated audio information", October 2007, <http://www.iso.org/iso/catalogue_detail?csnumber=44169>.

[ISO_14496] International Organization for Standardization, "ISO/IEC 14496-3:2009 Information technology -- Coding of audio-visual objects -- Part 3: Audio", 2009, <http://www.iso.org/iso/catalogue_detail?csnumber=53943>.

[ISO_8601] International Organization for Standardization, "ISO/IEC International Standard 8601:2004; Data elements and interchange formats -- Information interchange -- Representation of dates and times", December 2004, <http://www.iso.org/iso/catalogue_detail?csnumber=40874>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2964] Moore, K. and N. Freed, "Use of HTTP State Management", BCP 44, RFC 2964, October 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1113
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

[RFC6381] Gellens, R., Singer, D., and P. Frojdh, "The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types", RFC 6381, August 2011.

[US_ASCII] American National Standards Institute, "ANSI X3.4-1986, Information Systems -- Coded Character Sets 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)", December 1986.

12.2. Informative References

[ID3] ID3.org, "The ID3 audio file data tagging format", <http://www.id3.org/Developer_Information>.

[M3U] Nullsoft, Inc., "The M3U Playlist format, originally invented for the Winamp media player", <http://wikipedia.org/wiki/M3U>.

[SampleEnc] Apple Inc., "MPEG-2 Stream Encryption Format for HTTP Live Streaming", <https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/HLS_Sample_Encryption/>.

[UTI] Apple Inc., "Uniform Type Identifier", <http://developer.apple.com/library/ios/#documentation/general/conceptual/DevPedia-CocoaCore/UniformTypeIdentifier.html>.

[WebVTT] World Wide Web Consortium (W3C), "WebVTT Standard", September 2012, <http://dev.w3.org/html5/webvtt/>.

Authors' Addresses

Roger Pantos (editor)
Apple Inc.
Cupertino, California
United States

Email: http-live-streaming-review@group.apple.com

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1114
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

William May, Jr.
Apple Inc.
Cupertino, California
United States

Email: http-live-streaming-review@group.apple.com

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1115
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Informational
Internet-Draft
Intended status: Informational
Expires: October 2, 2011

R. Pantos, Ed.
W. May
Apple Inc.

March 31, 2011

HTTP Live Streaming
draft-pantos-http-live-streaming-06

Abstract

This document describes a protocol for transferring unbounded streams of multimedia data. It specifies the data format of the files and the actions to be taken by the server (sender) and the clients (receivers) of the streams. It describes version 3 of this protocol.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 2, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

This Informational Internet Draft is submitted as an RFC Editor

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1116
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Contribution and/or non-IETF Document (not as a Contribution, IETF Contribution, nor IETF Document) in accordance with BCP 78 and BCP 79.

Table of Contents

1. Introduction 4
2. Summary 4
3. The Playlist file 4
3.1. Introduction 4
3.2. Attribute Lists 5
3.3. New Tags 7
3.3.1. EXT-X-TARGETDURATION 7
3.3.2. EXT-X-MEDIA-SEQUENCE 7
3.3.3. EXT-X-KEY 7
3.3.4. EXT-X-PROGRAM-DATE-TIME 8
3.3.5. EXT-X-ALLOW-CACHE 8
3.3.6. EXT-X-PLAYLIST-TYPE 9
3.3.7. EXT-X-ENDLIST 9
3.3.8. EXT-X-STREAM-INF 9
3.3.9. EXT-X-DISCONTINUITY 10
3.3.10. EXT-X-VERSION 10
4. Media files 11
5. Key files 11
5.1. Introduction 12
5.2. IV for AES-128 12
6. Client/Server Actions 12
6.1. Introduction 12
6.2. Server Process 12
6.2.1. Introduction 12
6.2.2. Sliding Window Playlists 14
6.2.3. Encrypting media files 15
6.2.4. Providing variant streams 15
6.3. Client Process 16
6.3.1. Introduction 16
6.3.2. Loading the Playlist file 16
6.3.3. Playing the Playlist file 17
6.3.4. Reloading the Playlist file 18
6.3.5. Determining the next file to load 18
6.3.6. Decrypting encrypted media files 19
7. Protocol version compatibility 19
8. Examples 19
8.1. Introduction 19
8.2. Simple Playlist file 20
8.3. Sliding Window Playlist, using HTTPS 20
8.4. Playlist file with encrypted media files 20

Internet-Draft HTTP Live Streaming March 2011

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1117
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8.5. Variant Playlist file 21
9. Contributors 21
10. IANA Considerations 21
11. Security Considerations 22
12. References 23
12.1. Normative References 23
12.2. Informative References 24
Authors' Addresses 24

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1118
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1. Introduction

This document describes a protocol for transferring unbounded streams of multimedia data. The protocol supports the encryption of media data and the provision of alternate versions (e.g. bitrates) of a stream. Media data can be transferred soon after it is created, allowing it to be played in near real-time. Data is usually carried over HTTP [RFC2616].

External references that describe related standards such as HTTP are listed in Section 11.

2. Summary

A multimedia presentation is specified by a URI [RFC3986] to a Playlist file, which is an ordered list of media URIs and informational tags. Each media URI refers to a media file which is a segment of a single contiguous stream.

To play the stream, the client first obtains the Playlist file and then obtains and plays each media file in the Playlist. It reloads the Playlist file as described in this document to discover additional segments.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. The Playlist file

3.1. Introduction

Playlists MUST be Extended M3U Playlist files [M3U]. This document extends the M3U file format by defining additional tags.

An M3U Playlist is a text file that consists of individual lines. Lines are terminated by either a single LF character or a CR character followed by an LF character. Each line is a URI, a blank, or starts with the comment character '#'. Blank lines are ignored. White space MUST NOT be present, except for elements in which it is explicitly specified.

A URI line identifies a media file or a variant Playlist file (see Section 3.3.8).

URIs MAY be relative. A relative URI MUST be resolved against the

URI of the Playlist file that contains it.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1119
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Lines that start with the comment character '#' are either comments or tags. Tags begin with #EXT. All other lines that begin with '#' are comments and SHOULD be ignored.

The duration of a Playlist file is the sum of the durations of the media files within it.

M3U Playlist files whose names end in .m3u8 and/or have the HTTP Content-Type "application/vnd.apple.mpegurl" are encoded in UTF-8 [RFC3629]. Files whose names end with .m3u and/or have the HTTP Content-Type [RFC2616] "audio/mpegurl" are encoded in US-ASCII [US_ASCII].

Playlist files MUST have names that end in .m3u8 and/or have the Content-Type "application/vnd.apple.mpegurl" (if transferred over HTTP), or have names that end in .m3u and/or have the HTTP Content-Type type "audio/mpegurl" (for compatibility).

The Extended M3U file format defines two tags: EXTM3U and EXTINF. An Extended M3U file is distinguished from a basic M3U file by its first line, which MUST be #EXTM3U.

EXTINF is a record marker that describes the media file identified by the URI that follows it. Each media file URI MUST be preceded by an EXTINF tag. Its format is:

#EXTINF:<duration>,<title>

"duration" is an integer or floating-point number in decimal positional notation that specifies the duration of the media file in seconds. Integer durations SHOULD be rounded to the nearest integer. Durations MUST be integers if the protocol version of the Playlist file is less than 3. The remainder of the line following the comma is the title of the media file, which is an optional human-readable informative title of the media segment.

This document defines the following new tags: EXT-X-TARGETDURATION, EXT-X-MEDIA-SEQUENCE, EXT-X-KEY, EXT-X-PROGRAM-DATE-TIME, EXT-X-ALLOW-CACHE, EXT-X-PLAYLIST-TYPE, EXT-X-STREAM-INF, EXT-X-ENDLIST, EXT-X-DISCONTINUITY, and EXT-X-VERSION.

3.2. Attribute Lists

Certain extended M3U tags have values which are Attribute Lists. An Attribute List is a comma-separated list of attribute/value pairs with no whitespace.

An attribute/value pair has the following syntax:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1120
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

AttributeName=AttributeValue

An AttributeName is an unquoted string containing characters from the set [A-Z].

An AttributeValue is one of the following:

- decimal-integer: an unquoted string of characters from the set [0-9] expressing an integer in base-10 arithmetic.

- hexadecimal-integer: an unquoted string of characters from the set [0-9] and [A-F] that is prefixed with 0x or 0X and which expresses an integer in base-16 arithmetic.

- decimal-floating-point: an unquoted string of characters from the set [0-9] and '.' which expresses a floating-point number in decimal positional notation.

- quoted-string: a string of characters within a pair of double-quotes ("). The set of characters allowed in the string and any rules for escaping special characters are specified by the Attribute definition, but any double-quote (") character and any carriage-return or linefeed will always be replaced by an escape sequence.

- enumerated-string: an unquoted character string from a set which is explicitly defined by the Attribute. An enumerated-string will never contain double-quotes ("), commas (,), or whitespace.

- decimal-resolution: two decimal-integers separated by the "x" character, indicating horizontal and vertical pixel dimensions.

The type of the AttributeValue for a given AttributeName is specified by the Attribute definition.

A given AttributeName MUST NOT appear more than once in a given Attribute List.

An Attribute/value pair with an unrecognized AttributeName MUST be ignored by the client.

Attribute/value pairs of type enumerated-string that contain unrecognized values SHOULD be ignored by the client.

3.3. New Tags

3.3.1. EXT-X-TARGETDURATION

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1121
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The EXT-X-TARGETDURATION tag specifies the maximum media file duration. The EXTINF duration of each media file in the Playlist file MUST be less than or equal to the target duration. This tag MUST appear once in the Playlist file. Its format is:

#EXT-X-TARGETDURATION:<s>

where s is an integer indicating the target duration in seconds.

3.3.2. EXT-X-MEDIA-SEQUENCE

Each media file URI in a Playlist has a unique integer sequence number. The sequence number of a URI is equal to the sequence number of the URI that preceded it plus one. The EXT-X-MEDIA-SEQUENCE tag indicates the sequence number of the first URI that appears in a Playlist file. Its format is:

#EXT-X-MEDIA-SEQUENCE:<number>

A Playlist file MUST NOT contain more than one EXT-X-MEDIA-SEQUENCE tag. If the Playlist file does not contain an EXT-X-MEDIA-SEQUENCE tag then the sequence number of the first URI in the playlist SHALL be considered to be 0.

A media file's sequence number is not required to appear in its URI.

See Section 6.3.2 and Section 6.3.5 for information on handling the EXT-X-MEDIA-SEQUENCE tag.

3.3.3. EXT-X-KEY

Media files MAY be encrypted. The EXT-X-KEY tag provides information necessary to decrypt media files that follow it. Its format is:

#EXT-X-KEY:<attribute-list>

The following attributes are defined:

The METHOD attribute specifies the encryption method. It is of type enumerated-string. Each EXT-X-KEY tag MUST contain a METHOD attribute.

Two methods are defined: NONE and AES-128.

An encryption method of NONE means that media files are not encrypted. If the encryption method is NONE, the URI and the IV attributes MUST NOT be present.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1122
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

An encryption method of AES-128 means that media files are encrypted using the Advanced Encryption Standard [AES_128] with a 128-bit key and PKCS7 padding [RFC5652]. If the encryption method is AES-128, the URI attribute MUST be present. The IV attribute MAY be present; see Section 5.2.

The URI attribute specifies how to obtain the key. Its value is a quoted-string that contains a URI [RFC3986] for the key.

The IV attribute, if present, specifies the Initialization Vector to be used with the key. Its value is a hexadecimal-integer. The IV attribute appeared in protocol version 2.

A new EXT-X-KEY supersedes any prior EXT-X-KEY.

If the Playlist file does not contain an EXT-X-KEY tag then media files are not encrypted.

See Section 5 for the format of the key file, and Section 5.2, Section 6.2.3 and Section 6.3.6 for additional information on media file encryption.

3.3.4. EXT-X-PROGRAM-DATE-TIME

The EXT-X-PROGRAM-DATE-TIME tag associates the beginning of the next media file with an absolute date and/or time. The date/time representation is ISO/IEC 8601:2004 [ISO_8601] and SHOULD indicate a time zone. For example:

#EXT-X-PROGRAM-DATE-TIME:<YYYY-MM-DDThh:mm:ssZ>

See Section 6.2.1 and Section 6.3.3 for more information on the EXT-X-PROGRAM-DATE-TIME tag.

3.3.5. EXT-X-ALLOW-CACHE

The EXT-X-ALLOW-CACHE tag indicates whether the client MAY or MUST NOT cache downloaded media files for later replay. It MAY occur anywhere in the Playlist file; it MUST NOT occur more than once. The EXT-X-ALLOW-CACHE tag applies to all segments in the playlist. Its format is:

#EXT-X-ALLOW-CACHE:<YES|NO>

See Section 6.3.3 for more information on the EXT-X-ALLOW-CACHE tag.

3.3.6. EXT-X-PLAYLIST-TYPE

The EXT-X-PLAYLIST-TYPE tag provides mutability information about the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1123
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Playlist file. It is optional. Its format is:

#EXT-X-PLAYLIST-TYPE:<EVENT|VOD>

Section 6.2.1 defines the implications of the EXT-X-PLAYLIST-TYPE tag.

3.3.7. EXT-X-ENDLIST

The EXT-X-ENDLIST tag indicates that no more media files will be added to the Playlist file. It MAY occur anywhere in the Playlist file; it MUST NOT occur more than once. Its format is:

#EXT-X-ENDLIST

3.3.8. EXT-X-STREAM-INF

The EXT-X-STREAM-INF tag indicates that the next URI in the Playlist file identifies another Playlist file. Its format is:

#EXT-X-STREAM-INF:<attribute-list>
<URI>

The following attributes are defined:

BANDWIDTH

The value is a decimal-integer of bits per second. It MUST be an upper bound of the overall bitrate of each media file, calculated to include container overhead, that appears or will appear in the Playlist.

Every EXT-X-STREAM-INF tag MUST include the BANDWIDTH attribute.

PROGRAM-ID

The value is a decimal-integer that uniquely identifies a particular presentation within the scope of the Playlist file.

A Playlist file MAY contain multiple EXT-X-STREAM-INF tags with the same PROGRAM-ID to identify different encodings of the same presentation. These variant playlists MAY contain additional EXT-X-STREAM-INF tags.

CODECS

The value is a quoted-string containing a comma-separated list of formats, where each format specifies a media sample type that is present in a media file in the Playlist file. Valid format identifiers are those in the ISO File Format Name Space defined by

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1124
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RFC 4281 [RFC4281].

Every EXT-X-STREAM-INF tag SHOULD include a CODECS attribute.

RESOLUTION

The value is a decimal-resolution describing the approximate encoded horizontal and vertical resolution of video within the stream.

3.3.9. EXT-X-DISCONTINUITY

The EXT-X-DISCONTINUITY tag indicates an encoding discontinuity between the media file that follows it and the one that preceded it. The set of characteristics that MAY change is:

- file format
- number and type of tracks
- encoding parameters
- encoding sequence
- timestamp sequence

Its format is:

#EXT-X-DISCONTINUITY

See Section 4, Section 6.2.1, and Section 6.3.3 for more information about the EXT-X-DISCONTINUITY tag.

3.3.10. EXT-X-VERSION

The EXT-X-VERSION tag indicates the compatibility version of the Playlist file. The Playlist file, its associated media, and its server MUST comply with all provisions of the most-recent version of this document describing the protocol version indicated by the tag value.

Its format is:

#EXT-X-VERSION:<n>

where n is an integer indicating the protocol version.

A Playlist file MUST NOT contain more than one EXT-X-VERSION tag. A

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1125
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Playlist file that does not contain an EXT-X-VERSION tag MUST comply with version 1 of this protocol.

4. Media files

Each media file URI in a Playlist file MUST identify a media file which is a segment of the overall presentation. Each media file MUST be formatted as an MPEG-2 Transport Stream or an MPEG-2 audio elementary stream [ISO_13818].

Transport Stream files MUST contain a single MPEG-2 Program. There SHOULD be a Program Association Table and a Program Map Table at the start of each file. A file that contains video SHOULD have at least one key frame and enough information to completely initialize a video decoder.

A media file in a Playlist MUST be the continuation of the encoded stream at the end of the media file with the previous sequence number unless it was the first media file ever to appear in the Playlist file or it is prefixed by an EXT-X-DISCONTINUITY tag.

Clients SHOULD be prepared to handle multiple tracks of a particular type (e.g. audio or video). A client with no other preference SHOULD choose the one with the lowest numerical PID that it can play.

Clients MUST ignore private streams inside Transport Streams that they do not recognize.

The encoding parameters for samples within a stream inside a media file and between corresponding streams across multiple media files SHOULD remain consistent. However clients SHOULD deal with encoding changes as they are encountered, for example by scaling video content to accommodate a resolution change.

5. Key files

5.1. Introduction

An EXT-X-KEY tag with the URI attribute identifies a Key file. A Key file contains the cipher key that MUST be used to decrypt subsequent media files in the Playlist.

The AES-128 encryption method uses 16-octet keys. The format of the Key file is simply a packed array of these 16 octets in binary

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1126
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

format.

5.2. IV for AES-128

128-bit AES requires the same 16-octet Initialization Vector (IV) to be supplied when encrypting and decrypting. Varying this IV increases the strength of the cipher.

If the EXT-X-KEY tag has the IV attribute, implementations MUST use the attribute value as the IV when encrypting or decrypting with that key. The value MUST be interpreted as a 128-bit hexadecimal number and MUST be prefixed with 0x or 0X.

If the EXT-X-KEY tag does not have the IV attribute, implementations MUST use the sequence number of the media file as the IV when encrypting or decrypting that media file. The big-endian binary representation of the sequence number SHALL be placed in a 16-octet buffer and padded (on the left) with zeros.

6. Client/Server Actions

6.1. Introduction

This section describes how the server generates the Playlist and media files and how the client should download and play them.

6.2. Server Process

6.2.1. Introduction

The production of the MPEG-2 stream is outside the scope of this document, which simply presumes a source of a continuous stream containing the presentation.

The server MUST divide the stream into individual media files whose duration is less than or equal to a constant target duration. The server SHOULD attempt to divide the stream at points that support effective decode of individual media files, e.g. on packet and key frame boundaries.

The server MUST create a URI for each media file that will allow its clients to obtain the file.

The server MUST create a Playlist file. The Playlist file MUST conform to the format described in Section 3. A URI for each media file that the server wishes to make available MUST appear in the Playlist in the order in which it is to be played. The entire media file MUST be available to clients if its URI is in the Playlist file.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1127
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The Playlist file MUST contain an EXT-X-TARGETDURATION tag. Its value MUST be equal to or greater than the EXTINF value of any media file that appears or will appear in the Playlist file. Its value MUST NOT change. A typical target duration is 10 seconds.

The Playlist file SHOULD contain one EXT-X-VERSION tag which indicates the compatibility version of the stream. Its value MUST be the lowest protocol version with which the server, Playlist file, and associated media files all comply.

The server MUST create a URI for the Playlist file that will allow its clients to obtain the file.

If the Playlist file is distributed by HTTP, the server SHOULD support client requests to use the "gzip" Content-Encoding.

Changes to the Playlist file MUST be made atomically from the point of view of the clients.

The server MUST NOT change the Playlist file, except to:

- Append lines to it (Section 6.2.1).
- Remove media file URIs from the Playlist in the order that they appear, along with any tags that apply only to those media files (Section 6.2.2).
- Change the value of the EXT-X-MEDIA-SEQUENCE tag (Section 6.2.2).
- Add or remove EXT-X-STREAM-INF tags (Section 6.2.4). Note that clients are not required to reload variant Playlist files, so changing them may not have immediate effect.
- Add an EXT-X-ENDLIST tag to the Playlist (Section 6.2.1).

Furthermore, the Playlist file MAY contain an EXT-X-PLAYLIST-TYPE tag with a value of either EVENT or VOD. If the tag is present and has a value of EVENT, the server MUST NOT change or delete any part of the Playlist file (although it MAY append lines to it). If the tag is

present and has a value of VOD, the Playlist file MUST NOT change.

Every media file URI in a Playlist MUST be prefixed with an EXTINF tag indicating the duration of the media file.

The server MAY associate an absolute date and time with a media file by prefixing its URI with an EXT-X-PROGRAM-DATE-TIME tag. The value of the date and time provides an informative mapping of the timeline of the media to an appropriate wall-clock time, which may be used as a basis for seeking, for display, or for other purposes. If a server

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1128
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

provides this mapping, it SHOULD place an EXT-X-PROGRAM-DATE-TIME tag after every EXT-X-DISCONTINUITY tag in the Playlist file.

If the Playlist contains the final media file of the presentation then the Playlist file MUST contain the EXT-X-ENDLIST tag.

If the Playlist does not contain the EXT-X-ENDLIST tag, the server MUST make a new version of the Playlist file available that contains at least one new media file URI. It MUST be made available relative to the time that the previous version of the Playlist file was made available: no earlier than one-half the target duration after that time, and no later than 1.5 times the target duration after that time.

If the server wishes to remove an entire presentation, it MUST make the Playlist file unavailable to clients. It SHOULD ensure that all media files in the Playlist file remain available to clients for at least the duration of the Playlist file at the time of removal.

6.2.2. Sliding Window Playlists

The server MAY limit the availability of media files to those which have been most recently added to the Playlist. To do so the Playlist file MUST ALWAYS contain exactly one EXT-X-MEDIA-SEQUENCE tag. Its value MUST be incremented by 1 for every media file URI that is removed from the Playlist file.

Media file URIs MUST be removed from the Playlist file in the order in which they were added.

The server MUST NOT remove a media file URI from the Playlist file if the duration of the Playlist file minus the duration of the media file is less than three times the target duration.

When the server removes a media file URI from the Playlist, the media file SHOULD remain available to clients for a period of time equal to the duration of the media file plus the duration of the longest Playlist file in which the media file has appeared.

If a server plans to remove a media file after it is delivered to clients over HTTP, it SHOULD ensure that the HTTP response contains an Expires header that reflects the planned time-to-live.

6.2.3. Encrypting media files

If media files are to be encrypted the server MUST define a URI which will allow authorized clients to obtain a Key file containing a decryption key. The Key file MUST conform to the format described in Section 5.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1129
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The server MAY set the HTTP Expires header in the key response to indicate that the key may be cached.

If the encryption METHOD is AES-128, AES-128 CBC encryption SHALL be applied to individual media files. The entire file MUST be encrypted. Cipher Block Chaining MUST NOT be applied across media files. The IV used for encryption MUST be either the sequence number of the media file or the value of the IV attribute of the EXT-X-KEY tag, as described in Section 5.2.

The server MUST encrypt every media file in a Playlist using the method and other attributes specified by the EXT-X-KEY tag that most immediately precedes its URI in the Playlist file. Media files preceded by an EXT-X-KEY tag whose METHOD is NONE, or not preceded by any EXT-X-KEY tag, MUST NOT be encrypted.

The server MUST NOT remove an EXT-X-KEY tag from the Playlist file if the Playlist file contains a URI to a media file encrypted with that key.

6.2.4. Providing variant streams

A server MAY offer multiple Playlist files to provide different encodings of the same presentation. If it does so it SHOULD provide a variant Playlist file that lists each variant stream to allow clients to switch between encodings dynamically.

Variant Playlists MUST contain an EXT-X-STREAM-INF tag for each variant stream. Each EXT-X-STREAM-INF tag for the same presentation MUST have the same PROGRAM-ID attribute value. The PROGRAM-ID value for each presentation MUST be unique within the variant Playlist.

If an EXT-X-STREAM-INF tag contains the CODECS attribute, the attribute value MUST include every format defined by [RFC4281] that is present in any media file that appears or will appear in the Playlist file.

The server MUST meet the following constraints when producing variant streams:

- Each variant stream MUST present the same content, including stream discontinuities.

- Each variant Playlist file MUST have the same target duration.

- Content that appears in one variant Playlist file but not in another MUST appear either at the beginning or at the end of the Playlist file and MUST NOT be longer than the target duration.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1130
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Matching content in variant streams MUST have matching timestamps. This allows clients to synchronize the streams.

Elementary Audio Stream files MUST signal the timestamp of the first sample in the file by prepending an ID3 PRIV tag [ID3] with an owner identifier of "com.apple.streaming.transportStreamTimestamp". The binary data MUST be a 33-bit MPEG-2 Program Elementary Stream timestamp expressed as a big-endian eight-octet number, with the upper 31 bits set to zero.

In addition, all variant streams SHOULD contain the same encoded audio bitstream. This allows clients to switch between streams without audible glitching.

6.3. Client Process

6.3.1. Introduction

How the client obtains the URI to the Playlist file is outside the scope of this document; it is presumed to have done so.

The client MUST obtain the Playlist file from the URI. If the Playlist file so obtained is a variant Playlist, the client MUST obtain the Playlist file from the variant Playlist.

This document does not specify the treatment of variant streams by clients.

6.3.2. Loading the Playlist file

Every time a Playlist file is loaded or reloaded from the Playlist URI:

The client MUST ensure that the Playlist file begins with the EXTM3U tag and that the EXT-X-VERSION tag, if present, specifies a

protocol version supported by the client; if not, the client MUST NOT attempt to use the Playlist.

The client SHOULD ignore any tags and attributes it does not recognize.

The client MUST determine the next media file to load as described in Section 6.3.5.

If the Playlist contains the EXT-X-MEDIA-SEQUENCE tag, the client SHOULD assume that each media file in it will become unavailable at the time that the Playlist file was loaded plus the duration of the Playlist file. The duration of a Playlist file is the sum of the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1131
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

durations of the media files within it.

6.3.3. Playing the Playlist file

The client SHALL choose which media file to play first from the Playlist when playback starts. If the EXT-X-ENDLIST tag is not present and the client intends to play the media regularly (i.e. in playlist order at the nominal playback rate), the client SHOULD NOT choose a file which starts less than three target durations from the end of the Playlist file. Doing so can trigger playback stalls.

To achieve regular playback, media files MUST be played in the order that they appear in the Playlist file. The client MAY present the available media in any way it wishes, including regular playback, random access, and trick modes.

The client MUST be prepared to reset its parser(s) and decoder(s) before playing a media file that is preceded by an EXT-X-DISCONTINUITY tag.

The client SHOULD attempt to load media files in advance of when they will be required for uninterrupted playback to compensate for temporary variations in latency and throughput.

If the Playlist file contains the EXT-X-ALLOW-CACHE tag and its value is NO, the client MUST NOT cache downloaded media files after they have been played. Otherwise the client MAY cache downloaded media files indefinitely for later replay.

The client MAY use the value of the EXT-X-PROGRAM-DATE-TIME tag to display the program origination time to the user. If the value includes time zone information the client SHALL take it into account, but if it does not the client MUST NOT infer an originating time zone.

The client MUST NOT depend upon the correctness or the consistency of the value of the EXT-X-PROGRAM-DATE-TIME tag.

6.3.4. Reloading the Playlist file

The client MUST periodically reload the Playlist file unless it contains the EXT-X-ENDLIST tag.

However the client MUST NOT attempt to reload the Playlist file more frequently than specified by this section.

When a client loads a Playlist file for the first time or reloads a Playlist file and finds that it has changed since the last time it was loaded, the client MUST wait for a period of time before

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1132
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

attempting to reload the Playlist file again. This period is called the initial minimum reload delay. It is measured from the time that the client began loading the Playlist file.

The initial minimum reload delay is the duration of the last media file in the Playlist. Media file duration is specified by the EXTINF tag.

If the client reloads a Playlist file and finds that it has not changed then it MUST wait for a period of time before retrying. The minimum delay is a multiple of the target duration. This multiple is 0.5 for the first attempt, 1.5 for the second, and 3.0 thereafter.

In order to reduce server load, the client SHOULD NOT reload the Playlist files of variant streams that are not currently being played. If it decides to switch playback to a different variant, it SHOULD stop reloading the Playlist of the old variant and begin loading the Playlist of the new variant. It can use the EXTINF durations and the constraints in Section 6.2.4 to determine the approximate location of corresponding media. Once media from the new variant has been loaded, the timestamps in the media files can be used to synchronize the old and new timelines precisely.

6.3.5. Determining the next file to load

The client MUST examine the Playlist file every time it is loaded or reloaded to determine the next media file to load.

The first file to load MUST be the file that the client has chosen to play first, as described in Section 6.3.3.

If the first file to be played has been loaded and the Playlist file does not contain the EXT-X-MEDIA-SEQUENCE tag then the client MUST verify that the current Playlist file contains the URI of the last

loaded media file at the offset it was originally found at, halting playback if it does not. The next media file to load MUST be the first media file URI following the last-loaded URI in the Playlist.

If the first file to be played has been loaded and the Playlist file contains the EXT-X-MEDIA-SEQUENCE tag then the next media file to load SHALL be the one with the lowest sequence number that is greater than the sequence number of the last media file loaded.

6.3.6. Decrypting encrypted media files

If a Playlist file contains an EXT-X-KEY tag that specifies a Key file URI, the client MUST obtain that key file and use the key inside it to decrypt all media files following the EXT-X-KEY tag until another EXT-X-KEY tag is encountered.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1133
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

If the encryption METHOD is AES-128, AES-128 CBC decryption SHALL be applied to individual media files. The entire file MUST be decrypted. Cipher Block Chaining MUST NOT be applied across media files. The IV used for decryption MUST be either the sequence number of the media file or the value of the IV attribute of the EXT-X-KEY tag, as described in Section 5.2.

If the encryption METHOD is NONE, the client MUST treat all media files following the EXT-X-KEY tag as cleartext (not encrypted) until another EXT-X-KEY tag is encountered.

7. Protocol version compatibility

Clients and servers MUST implement protocol version 2 or higher to use:

- The IV attribute of the EXT-X-KEY tag.

Clients and servers MUST implement protocol version 3 or higher to use:

- Floating-point EXTINF duration values.

8. Examples

8.1. Introduction

This section contains several example Playlist files.

8.2. Simple Playlist file

```
#EXTM3U
#EXT-X-TARGETDURATION:5220
#EXTINF:5220,
http://media.example.com/entire.ts
#EXT-X-ENDLIST
```

8.3. Sliding Window Playlist, using HTTPS

```
#EXTM3U
#EXT-X-TARGETDURATION:8
#EXT-X-MEDIA-SEQUENCE:2680

#EXTINF:8,
https://priv.example.com/fileSequence2680.ts
```

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1134
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   #EXTINF:8,
   https://priv.example.com/fileSequence2681.ts
   #EXTINF:8,
   https://priv.example.com/fileSequence2682.ts
```

8.4. Playlist file with encrypted media files

```
   #EXTM3U
   #EXT-X-MEDIA-SEQUENCE:7794
   #EXT-X-TARGETDURATION:15

   #EXT-X-KEY:METHOD=AES-128,URI="https://priv.example.com/key.php?r=52"

   #EXTINF:15,
   http://media.example.com/fileSequence52-1.ts
   #EXTINF:15,
   http://media.example.com/fileSequence52-2.ts
   #EXTINF:15,
   http://media.example.com/fileSequence52-3.ts

   #EXT-X-KEY:METHOD=AES-128,URI="https://priv.example.com/key.php?r=53"

   #EXTINF:15,
   http://media.example.com/fileSequence53-1.ts
```


8.5. Variant Playlist file

```
   #EXTM3U
   #EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1280000
   http://example.com/low.m3u8
   #EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2560000
   http://example.com/mid.m3u8
   #EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
   http://example.com/hi.m3u8
   #EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=65000,CODECS="mp4a.40.5"
   http://example.com/audio-only.m3u8
```

9. Contributors

Significant contributions to the design of this protocol were made by Jim Batson, David Biderman, Bill May, Roger Pantos, and Alan Tseng.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1135
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

10. IANA Considerations

This memo requests that the following MIME type [RFC2046] be registered with the IANA:

Type name: "application"

Subtype name: "vnd.apple.mpegurl"

Required parameters: (none)

Optional parameters: (none)

Encoding considerations: encoded as text. See Section 3 for more information.

Security considerations: See Section 11.

Compression: this media type does not employ compression.

Interoperability considerations: There are no byte-ordering issues, since files are 7- or 8-bit text. Applications could encounter unrecognized tags, which SHOULD be ignored.

Published specification: see Section 3.

Applications that use this media type: Multimedia applications such as the iPhone media player (OS 3.0) and QuickTime Player in Mac OS X Snow Leopard.

Additional information: files begin with the magic number #EXTM3U. Filenames normally end with .m3u8 or .m3u (see Section 3). No Macintosh file type codes have been registered.

Person & email address to contact for further information: David Singer, singer AT apple.com.

Intended usage: LIMITED USE

Restrictions on usage: (none)

Author: Roger Pantos

Change Controller: David Singer

11. Security Considerations

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1136
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Since the protocol generally uses HTTP to transfer data, most of the same security considerations apply. See section 15 of RFC 2616 [RFC2616].

Media file parsers are typically subject to "fuzzing" attacks. Clients SHOULD take care when parsing files received from a server so that non-compliant files are rejected.

Playlist files contain URIs, which clients will use to make network requests of arbitrary entities. Clients SHOULD range-check responses to prevent buffer overflows. See also the Security Considerations section of RFC 3986 [RFC3986].

Clients SHOULD load resources identified by URI lazily to avoid contributing to denial-of-service attacks.

HTTP requests often include session state ("cookies"), which may contain private user data. Implementations MUST follow cookie restriction and expiry rules specified by RFC 2965 [RFC2965]. See also the Security Considerations section of RFC 2965, and RFC 2964 [RFC2964].

Encryption keys are specified by URI. The delivery of these keys SHOULD be secured by a mechanism such as HTTP over TLS [RFC5246] (formerly SSL) in conjunction with a secure realm or a session cookie.

12. References

12.1. Normative References

[AES_128] U.S. Department of Commerce/National Institute of Standards and Technology, "Advanced Encryption Standard (AES), FIPS PUB 197", November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[ISO_13818] International Organization for Standardization, "ISO/IEC International Standard 13818; Generic coding of moving pictures and associated audio information", October 2007, <http://www.iso.org/iso/catalogue_detail?csnumber=44169>.

[ISO_8601] International Organization for Standardization, "ISO/IEC International Standard 8601:2004; Data elements and interchange formats -- Information interchange -- Representation of dates and times", December 2004, <http://www.iso.org/iso/catalogue_detail?csnumber=40874>.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1137
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2964] Moore, K. and N. Freed, "Use of HTTP State Management", BCP 44, RFC 2964, October 2000.

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, October 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4281] Gellens, R., Singer, D., and P. Frojdh, "The Codecs Parameter for "Bucket" Media Types", RFC 4281, November 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[US_ASCII] American National Standards Institute, "ANSI X3.4-1986, Information Systems -- Coded Character Sets 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)", December 1986.

12.2. Informative References

[ID3] ID3.org, "The ID3 audio file data tagging format", <http://www.id3.org/Developer_Information>.

[M3U] Nullsoft, Inc., "The M3U Playlist format, originally invented for the Winamp media player", <http://wikipedia.org/wiki/M3U>.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1138
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Authors' Addresses

Roger Pantos (editor)
Apple Inc.
Cupertino, California
United States

Email: http-live-streaming-review@group.apple.com

William May, Jr.
Apple Inc.
Cupertino, California
United States

Email: http-live-streaming-review@group.apple.com

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1139
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Experimental Characterization of Home Wireless Networks and Design Implications

Konstantina Papagiannaki‡, Mark Yarvis†, W. Steven Conner†
‡: Intel Research Cambridge, †: Intel Corporation
e-mail: dina.papagiannaki@intel.com, mark.d.yarvis@intel.com, w.steven.conner@intel.com

Abstract— **Anecdotal evidence suggests that home wireless networks may be unpredictable despite their limited size. In this work, we deploy six-node wireless testbeds in three houses in the United States and the United Kingdom. We examine the quality of links in home wireless networks and the effect of (i) transmission rate, (ii) transmission power, (iii) node location, (iv) type of house, and (v) 802.11 technology. We provide empirical evidence suggesting that homes are challenging environments for wireless communication. Wireless links in the home are highly asymmetric and heavily influenced by precise node location, transmission power, and encoding rate, rather than physical distance between nodes. In our measurements, many links were unable to utilize the maximum transmission rate of the deployed 802.11 technology, and a few provided no connectivity at all. These results suggest that creating an AP-based topology with maximum coverage and throughput in this environment is challenging. Our findings have implications on the design of future home wireless networks and requirements for future wifi-enabled consumer electronic devices. We show that coverage and performance can be improved using a multi-hop topology, implying that mesh capabilities may actually be needed in consumer electronics for seamless connectivity across the home.**

I. Introduction

Wireless networks have become increasingly popular due to ease of deployment and low cost compared to wired networks. However, the transmission principles in wireless communications are dramatically different than those of wired networks. A recent study of wireless access points deployed over a metropolitan area demonstrates significant challenges to performance and connectivity [1]. Similarly, deployment of a wireless network in an enterprise environment, while relatively well understood, typically requires a site survey to engineer a network with proper coverage and capacity. Comparatively little is known about the properties of home wireless networks, beyond anecdotal evidence [2]. Even less is known about optimal design of such networks. Unfortunately, previous studies of campus/metropolitan-area networks and enterprise networks are hardly applicable to home networks, which tend to be much smaller (both in total size and contiguous space), have a single access point, are almost entirely indoors, and have no IT staff to perform site surveys. In this paper we consider factors that impact the design and performance of home wireless networks.

Before investigating network design, we first attempt to measure the characteristics of home wireless networks. A typical home wireless network consists of an access point (AP), several PCs, and increasingly, consumer electronic (CE) devices. Today a home wireless network is primarily used to provide access to a wired Internet connection; communication occurs to and from an AP. Future home wireless networks are expected to feature different types of traffic (such as multi-media streaming) which may require efficient communication between any two devices in the home. Such an environment can be enabled using 802.11 ad hoc mode, 802.11 Wireless Distribution System (WDS) frames, or the proposed 802.11e Direct Link Setup (DLS) capability [3]. An evaluation of typical link characteristics in the home environment is crucial to understanding the behavior of both traditional AP-based as well as future network topologies.

Using small networks of devices deployed in three homes, two in the United States and one in the United Kingdom, we study the properties of wireless links in home environments using a pure measurement approach. We examine the impact of transmission rate and transmission power on link quality, in terms of success rate and throughput. We show that despite the small size of home wireless networks, connectivity between any two wireless devices is not guaranteed or necessarily predictable, regardless of transmission power or rate. We also show that small changes in antenna orientation and node location can have a dramatic and unpredictable impact on the connectivity of the network. Our results span both 802.11a and 802.11b technologies and do not show strong correlation with the physical distance between nodes. These results suggest that a typical home user cannot depend on common sense alone in deploying a high-performance wireless network.

The above results have a direct impact on home wireless network design. Infrastructure mode wireless networks typically deployed in homes require one or more access points, through which all other nodes communicate. Our characterization of connectivity in the home suggests that most of the possible access point deployment locations fail to provide full coverage of high-performance connectivity throughout a typical home. To find the optimal location for access point deployment, a homeowner could resort to a site survey. Design of medium or large-scale wireless networks typically relies on a site survey or tools that model radio signal propagation, taking into account a detailed floor plan along with construction materials and placement of household objects [4], [5]. These solutions are far too costly and time-consuming to be applied in the home. In addition, aesthetics and the location at which the Internet enters the home are usually overriding concerns for the home user when selecting the location for an access point.

* Other names and brands may be claimed as the property of others.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1140
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

A home user is much more likely to put an access point in the corner of an office than the middle of a living room, even if the later location is known to provide optimal performance. A second option would be to deploy multiple access points. However, this approach typically requires the wired network to be extended to multiple points within the home. Thus, this option would eliminate the key advantage of home wireless networking: low installation cost. Our results suggest that more flexible topologies can provide a more appropriate alternative in home environments.

The measurements collected from our 6-node testbeds show that across all three houses there is always at least one pair of nodes that cannot directly communicate. Moreover, a substantial number of pairs cannot communicate at the highest rate supported by the deployed 802.11 technology. Using these measurements we study the ability of alternative topologies to alleviate these problems. More precisely, we study the impact of (i) AP, (ii) direct, and (iii) mesh topologies. We show that the AP topology typical in home networks today is highly sensitive to the location of the AP and in many cases leads to sub-optimal performance, due to the mandatory use of the AP as a relay. A direct topology eliminates the need to transmit each packet twice and also allows nearby nodes to utilize a high-rate channel encoding, which would not otherwise be possible if the access point is far away. However, in a direct topology not all nodes will be able to communicate with each other. On the other hand, a mesh topology can help nodes overcome the limitations of their environment, establishing either direct connections or multi-hop connections through neighboring nodes as appropriate. Our results show that solutions enabling mesh topologies [6] can help eliminate poor quality links in the home while establishing paths for the communication of nodes that are otherwise unreachable. To the best of our knowledge, our work is the first one to identify and quantify such challenges in home wireless communication and to demonstrate the need for mesh capabilities (e.g., IEEE 802.11s [7]) on wifi-enabled consumer electronics.

The remainder of the paper is structured as follows. In Section II we present the experimental methodology followed throughout the paper. In Section III we assess the quality of wireless links inside the house in terms of reachability. Throughput measurements are presented in Section IV. We study the impact of alternative topologies on the performance of a home wireless network in Section V. We present related work in Section VI and conclude in Section VII.

II. Experimental Environment

Our experiments are intended to assess the quality of wireless links in home environments. We evaluate three homes, two in the United States and one in the United Kingdom. High-level details of the different homes hosting our experiments can be found in Table I[1]. Our experiments are designed to investigate the impact of the following parameters:

- Type of house, e.g. size, construction material.
- Wireless technology used: 802.11a or 802.11b[2].

[1]In *ushome2*, the bottom floor was partially below grade.

[2]The wireless cards used also support 802.11g, which has not been reported upon in this paper due to the poor 802.11g support in the driver.

TABLE I
Description of homes used in experimental testbeds.

Label	Size (ft^2)	Construction	# Floors	# Nodes
ushome1	2,500	Wood	2	6
ushome2	2,600	Wood	3	6
ukhome1	1,500	Brick / steel	3	6

- Transmission power, denoted by *txpower*.
- Transmission rate, denoted by *txrate*.
- Node location.

A. Experimental Setup

We deploy six wireless nodes inside each home[3]. Nodes are located in different rooms, wherever computing or consumer electronic devices might be found in the home. For 802.11b experiments the nodes are small form-factor PCs with Netgear MA701 compact flash 802.11b wireless cards. The nodes run Linux kernel version 2.4.19 and the hostap driver [8]. For 802.11a experiments the nodes are laptops with NetGear WAG511 CardBus 802.11a/b/g cards running Linux kernel version 2.4.26 and the MIT madwifi-stripped driver [9]. All radios have omnidirectional antennas, and could be considered comparable to the radios that are likely to be integrated in future consumer electronics, e.g. cheap radios with basic functionality. Lastly, all three testbeds are homogeneous; each node consists of the exact same hardware to limit the impact of hardware peculiarities on the obtained results.

Our network setup is common among all experiments. All nodes utilize an unused frequency that is at least five channels away from the next occupied 802.11 frequency. To facilitate our experiments, we utilize the 802.11 Independent Basic Service Set (IBSS) mode, which allows all nodes to communicate directly. However, this configuration does not constrain the usefulness of our results to ad hoc (mesh) topologies, as will be described in Section V. Each node is instructed to run an experiment toward every other node in turn. Our experiments are designed to assess: (i) success/loss rate, and (ii) throughput under different combinations of *txrate* and *txpower*. We further alter the node location for specific experiments in order to quantify the impact of exact node location, antenna orientation, and obstacles. Experiments are carried out during the night to avoid interference from moving people and facilitate reproducibility. Except where explicitly stated, each result represents a single experimental run, due to the highly time consuming nature of our experiments. Instead, we rely on the validating runs presented in the following subsections to lend credence to our results.

1) Reachability: The reachability experiments assess link quality between each pair of nodes in the home network in terms of success/loss rate, and rely on a series of UDP probe packets sent from every node to every other node. Each probe packet lists the source node, as well as its number in the series. The size of the probe packet and the duration of each

[3]The nodes are denoted as node-2 through node-7 in the remainder of the paper. Node-1 controls each experiment, instructing each node in turn about the type and parameters of the experiment.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1141
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

sub-experiment are configurable. In all experiments, link-layer retransmissions were disabled, the probe size was 1472 bytes, and the duration of each sub-experiment was 60 seconds, with a frequency of one packet every 500 ms. Each individual wireless link is assessed independently, and no simultaneous transmissions take place inside the network.

2) Throughput: The throughput experiments were run to assess the quality of layer-3 communication within the home network. Throughput was assessed for both TCP and UDP protocols. For our throughput experiments we use the same basic methodology which relies on measurements from all pairings of the six nodes in the testbed. Throughput is measured by the *netperf* traffic generator using 1472 byte packets. Each node initiates a netperf connection to every other node (in turn) and measures the throughput achieved over a 60 second time interval. Unlike the reachability experiments, the throughput experiments are conducted with link layer retransmission enabled (maximum of 3 retries), which is likely to alleviate the effect of short term degradation in link quality.

B. Basic Methodology

Each reachability experiment quantifies the loss rate observed by each wireless link, as well as the sequence of successes and failures. We graphically present the obtained measurement matrix in Figure 1 as collected in *ushome2* when *txrate* is 2 Mbps and *txpower* is 30 mW. Every row in Figure 1 corresponds to probes sent from a specific source node. In each subplot, a bar denotes the successful reception of a probe by the destination node.

From Figure 1 we note that in *ushome2* and under the selected transmission power and rate, communication from node-2 to node-7 is extremely limited; most probe packets were lost. In addition, we see that while node-2 has a low delivery success rate to node-6, the quality of the link in the reverse direction is significantly better (nearly 100% success). Such link asymmetry has been reported in previous performance studies of wireless networks [1], [10] and was found to be quite common in the home environments studied in this paper. Throughput experiments simply output the rate achieved for each pair of nodes under the different *txrate*, *txpower* combinations, along with the respective packet loss rate.

Given the large amount of experimental data collected for this study and the number of different dimensions, we tag the experiments with the following 4-tuple: *home-layout-type-experiment*. The first tuple can be *ushome1*, *ushome2*, or *ukhome1*. The second tuple reflects the positions of the different nodes and is of the form *layout*1,2,.., etc. The third tuple denotes whether the experiment assesses *link* loss/success rate, *UDP*, or *TCP* throughput. The last tuple denotes the goal of the experiment, for instance in the next section we present results from our *validation* experiments.

C. Validation

To validate whether our experimental results represent actual link characteristics or a transient effect, we run two experiments with the exact same node deployment (denoted by *layout1*) and at the same time of day. We then compute

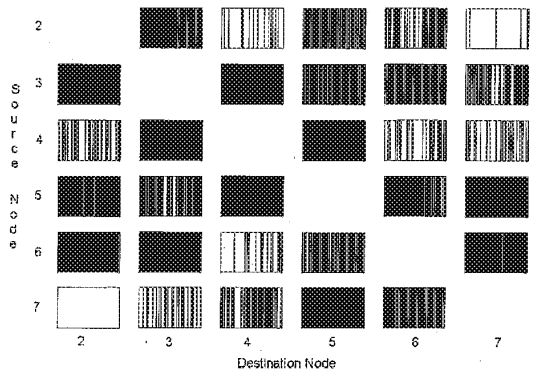


Fig. 1. Matrix of probe packets successfully delivered between each pair of nodes in *ushome2* at 30mW and 2Mbps (*ushome2 – layout1 – link – validation1*).

the loss and throughput rates observed for each wireless link under each test. We consider this an essential step of our measurement methodology, to identify whether a change in the collected measurements, across a specific experiment, reflects the effect of the studied parameter (e.g. *txrate* or *txpower*) and not the inherent variability in the measurements themselves.

1) Reachability: In Figure 2 we present the results obtained across two experiments with the same setup. Each experiment results in four subplots, where *txrate* is either 2 Mbps or 11 Mbps and *txpower* is either 30 mW or 1 mW. Each subplot contains the performance of individual wireless links in terms of their loss rate in each direction[4] (Figure 1 contains the source data presented in the lower left plot of Figure 2(a).). While not exactly identical, the performance shown in each subgraph of Figure 2(a) is similar to that of Figure 2(b). Links that are poor or asymmetric in one run, tend to also be poor or asymmetric in the next. Thus, network performance does not change significantly from one run to the next. We ran the same validation test in each home and found this result was easily reproducible.

To determine whether 60 seconds is sufficient to obtain an accurate view of the quality of the wireless link, we also measured links over a longer period of time. Using a transmission rate of 11 Mbps and a transmit power of 30 mW, we performed the same experiment in *ushome1* for a time span of 20 minutes (instead of 1 minute). We then compared the success rates derived using the entire time series with the success rates that would be estimated by the first 120 samples (i.e. 60 seconds). In Figure 3, each point represents the two success rate measurements for each unidirectional link. In each case, the success rate measured in 60 seconds was a reasonable estimate of the success rate over the 20 minute period. Thus, 60 seconds is long enough to assess the medium-term properties of each link under the tested conditions.

[4]An easy way to read the performance summaries in Figure 2 is to observe how busy they are; a busy plot implies poor performance, a mostly empty plot indicates no loss across the majority of node pairs. Similar summaries are used to capture throughput results, in which case absent bars indicate zero throughput along a particular link direction.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1142
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

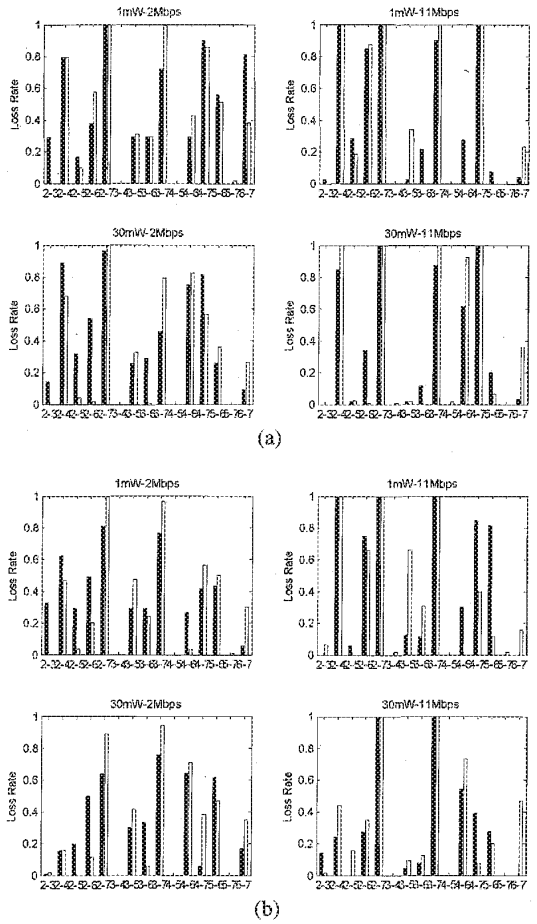


Fig. 2. Loss rates for each pair of nodes in experiments (a) $ushome2 - layout1 - link - validation1$, and (b) $ushome2 - layout1 - link - validation2$. Average loss from top-left to bottom-right: (a) 0.42, 0.37, 0.35, 0.35, (b) 0.33, 0.38, 0.29, 0.29.

We must also consider the effect of time of day on link performance. Recall that experiments were typically performed at night to avoid interference from household activity. To determine if results obtained at other times of day would vary significantly, we performed a 60-second link test for a single node pair in *ushome1* once per hour for 24 hours. As shown in Figure 4, while link quality may fluctuate somewhat with time, a "good" link tends to remain "good" (and a "bad" link remains "bad"), despite small deviations over time. To avoid any complications from time of day specific behavior, we tried to collect comparable data at the same time of day.

2) Throughput: The validation process for the throughput experiments involves the following steps. First, we compare the UDP results obtained across two different runs on two sequential days with the same node configuration. We found that results were easily reproducible (Figure 5). Second, we compare whether the throughput rates achieved agree with the success rates reported from the preceding reachability

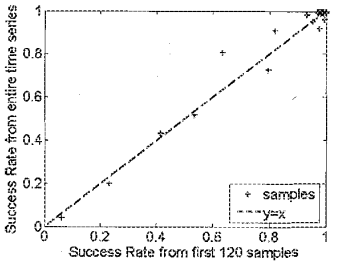


Fig. 3. Comparison of success rate results for 120 and 2400 sample lengths. The straight line is used as reference of equality (y=x) ($ushome1 - layout1 - link - long$, *txrate=11Mbps, txpower=30mW*).

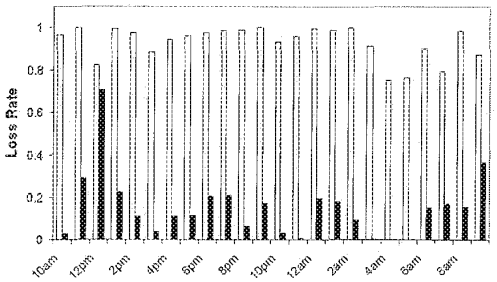


Fig. 4. Loss rate as a function of time of day for *ushome1* (*txpower=30mW, txrate=11Mbps*). First bar is node-4 to node-6, second bar is node-6 to node-4. ($ushome1 - layout1 - link - day$, *txrate=11Mbps, txpower=30mW*)

experiment. Indeed, Figure 6 confirms such a statement.

In Figure 6 we see that good quality links typically achieve the highest throughput supported, while bad links typically obtain low throughput. We note that the set of reachability measurements is taken before the set of throughput measurements, with a one hour time lag between the two measurements of a given link. Links with occasional losses may have variable performance. In addition, throughput experiments were carried out with link level transmission turned on (maximum of 3 retries), while reachability experiments were performed with no link layer retransmission. Throughput provides a more accurate picture of these losses, resulting in intermediate throughput values. From Figure 6, however, it is clear that high loss rate on any direction leads to very poor UDP throughput along that direction. In case of TCP throughput, if high loss is observed on the forward direction then TCP throughput is very low (link 3-6). If high loss rate is observed on the reverse direction, then TCP throughput is not zero but significantly smaller (links 2-3, 5-6, 5-6) than what could be achieved on a loss-free link on both directions. We note that such great sensitivity is not evident in our measurements when the transmission rate is 2 Mbps due to the robustness of the encoding scheme (also seen in Figure 5).

3) Experimental Setup: One last aspect of the experimental setup that needs to be validated has to do with the hardware itself. Wireless measurements are likely to be influenced by the hardware and software used. The results presented henceforth are specific to the platform we use. However,we must validate whether specific observed characteristics are an artifact of the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1143
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

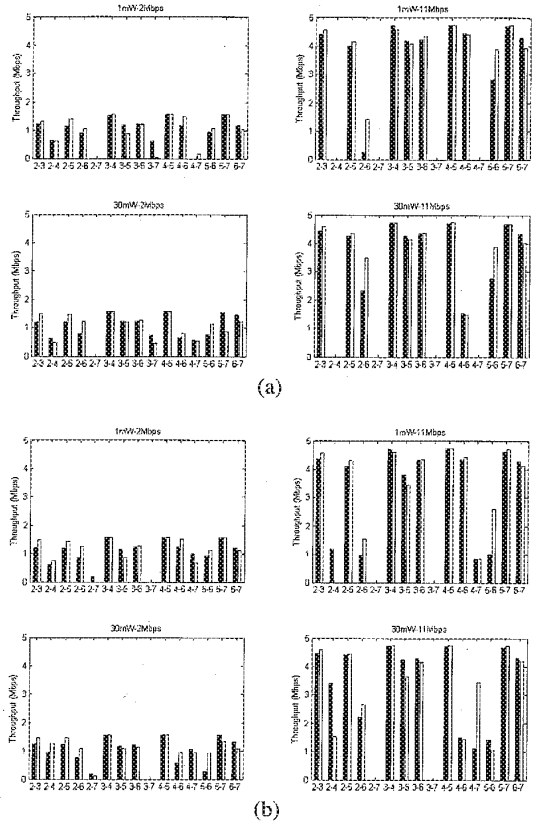


Fig. 5. UDP throughput achieved across validation runs, (a) *ushome2 – layout1 – udp – validation1*, (b) *ushome2 – layout1 – udp – validation2*. Average throughput in Mbps from top-left to bottom-right: (a) 1.01, 2.92, 1.01, 2.89, (b) 1.06, 2.92, 1.02, 3.02.

behavior of specific nodes. To discount such a possibility, all three testbeds comprise the exact same hardware and run the exact same versions of the respective software. Before we started the described experiments we had to replace specific nodes that appeared to mis-behave, sourcing all asymmetric links for instance. After each individual testbed was assessed to function properly, we rotated nodes and radios around the house (node-2 moved to the position of node-3, node-3 to the position of node-4, etc.), and observed whether there were specific trends in the results. Our findings validated that all nodes are equivalent in terms of quality.

III. Reachability Results

We now evaluate the home wireless environment along five dimensions: (i) *txrate*, (ii) *txpower*, (iii) node location, (iv) house type, (v) physical layer[5]. Due to space limitations, we present results for the most interesting experiments across homes and settings. All experimental results from all houses are available at [12].

[5]We have also looked at external interference (e.g., from a microwave oven) in a preliminary version of this work [11].

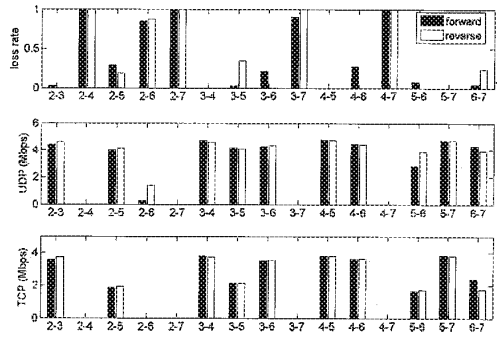


Fig. 6. Relationship between loss rate and UDP/TCP throughput (*ushome2 – layout – * – validation1*, *txrate=11Mbps*, *txpower=1mW*).

A. Overall characteristics of a home network

Using 802.11b radios, a full set of measurements like those presented in Figure 1 was collected for four combinations of transmission power and rate. In Figure 7(a) we present our results for all combinations from *ushome2*. The deployment of nodes in the individual homes is schematically shown in Figure 8. We refer to the initial layout of the nodes in each home as *layout1*.

As expected, in most cases link loss rates are higher when the encoding rate is higher and somewhat lower when the power level increases. While each home represents a small space, wireless connectivity is not always omnipresent. Across all rates and power levels, a large number of asymmetric links are present. In most experiments, at least one pair of nodes has greater than 30% loss. And, as seen in Figure 7(a), while the increase in transmission power improves some links, the overall problem is not eliminated. This initial set of experiments demonstrates that lossy links are likely to be found inside a home, and in some cases, loss cannot be eliminated by reducing the transmission rate or increasing the transmission power. On the other hand, such changes do not appear to affect the quality of links with low loss rates.

B. Small changes in antenna orientation and location

There are several reasons why particular node pairs may not be able to communicate. The location of the nodes and the orientation of their antennas impact the obstacles in their direct path, and thus multi-path fading and signal attenuation. To evaluate these effects in the home, we modify *layout1* by rotating all nodes 180°, such that their antennas face the opposite direction (such a change may have also slightly changed their exact location). We call this deployment *layout2*. We perform the same series of experiments on *layout2* and present the results in Figure 7(b).

We observe that a small change in node location and orientation can have a significant impact on link quality. The number of links with a loss rate above 50% in *layout2* is smaller than the one in *layout1*. Since the distance between nodes does not change significantly, and since the change observed between *layout1* and *layout2* is much greater than

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1144
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

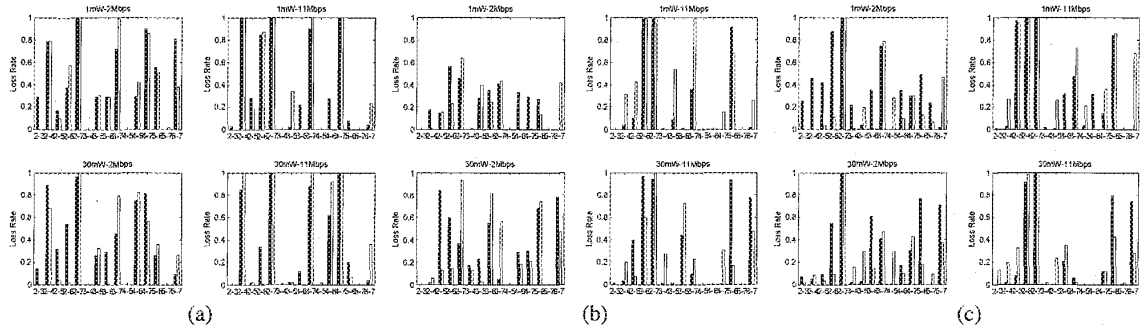


Fig. 7. Loss rate for node pairs for (a) *layout1*, (b) *layout2*, and (c) *layout3* in *ushome2* (*ushome2* $- * -$ *link* $-$ *placement*). Average loss from top-left to bottom-right: (a) 0.42, 0.37, 0.35, 0.35, (b) 0.19, 0.29, 0.31, 0.29, (c) 0.31, 0.38, 0.28, 0.28.

normal variation (Section II-C), exact node placement must be a key contributor to performance.

Our findings for *ushome2* are summarized in Figure 8(b). The leftmost figure denotes the node pairs that experience the worst connectivity (links with greater than 95% loss) in *layout1*. In the middle we identify links with the worst connectivity in *layout2*. Under the new configuration, the set of nodes that cannot communicate has changed. Similar findings were obtained for *ushome1* and *ukhome1* as shown in Figure 8(a) and (c), with *ushome1* experiencing the most dramatic changes in performance between *layout1* and *layout2*.

C. Large changes in node placement

The previous section considered the impact of small changes in node location. We now move each node in *ushome2* from its positions in *layout1* to a different location within the same room (rightmost plot of Figure 8(b)). We call this configuration *layout3*. Loss rates measured for *layout3* are shown in Figure 7(c). We observe that *layout3* leads to a more "concentrated" loss area than *layout1*.

The above results clearly demonstrate the challenges of home environments on the design and performance of home wireless networks. Node positioning has a dramatic impact on network connectivity, and "randomly" selecting the location of a node will not ensure its connectivity. Moreover, "randomly" selecting the location for an access point does not necessarily ensure a fully connected network. Consider use of an AP topology in our testbed, with one testbed node replaced by an access point. For the example of *layout3*, node locations 2, 5, 6 and 7 would not be good choices for an AP, as they would not have good connectivity to all other nodes.

D. The relationship between link quality and distance

In Section III-A we demonstrated that home wireless links tend to be highly asymmetric. The presence of asymmetry suggests a loose relationship between distance and link quality. In this section we look into this question in more detail.

Figure 9 presents the loss rate between node pairs for *layout2* in each home versus the distance between the nodes. Clearly there is no correlation between physical distance and wireless link quality in these home networks. This result holds across homes and across *txrate* and *txpower* settings.

Our results thus far demonstrate that the performance of a home wireless link tends to be most affected by the set of objects between the two endpoints, rather than physical distance or transmission power.

E. Comparison between home networks

Across homes, results differ substantially. Figure 10 presents loss rates for *layout1* across all three homes. In *layout1*, the largest home, *ushome2*, has the worst performance (Figure 10(b)), and the smallest home, *ukhome1* has the best performance, particularly at low transmit power (Figure 10(c)). Results for *ushome1* (Figure 10(a)) are significantly different from the results for *ushome2* even though they are similar in size. *Ushome1* and *ushome2* differ in that *ushome2* is a three-story building, while *ushome1* has two floors. Nonetheless, the impact of the number of floors is not evident for *ukhome1*, which allowed an almost fully connected wireless network. While some of the observations above could suggest that distance or size play a significant role in performance, the overall results presented here and in Section III-D demonstrate that loss rate cannot be predicted based on such features. The key parameter is precise node location and orientation, rather than home size or distance between nodes.

F. The impact of the physical layer: 802.11a

While the preceding data was collected using the IEEE 802.11b physical layer, other physical layers may possess different characteristics. In this section, we consider the performance of the IEEE 802.11a physical layer in the home. As described in Section II, we deploy laptops with 802.11a wireless cards in the same locations as the 802.11b nodes and perform the same series of connectivity experiments. Each experiment is completed with the same transmission power: 30 mW. We considered four different link encoding rates: 6 Mbps, 18 Mbps, 36 Mbps, and 54 Mbps. Two node deployments were used, where *layout1* is the initial deployment, and in *layout2* nodes are rotated by 180°. The loss rates in *ushome1* for *layout1* and *layout2* are reported in Figure 11[6].

[6]802.11a experiments were not possible in the UK home due to local regulations and limitations of the card's IBSS implementation.

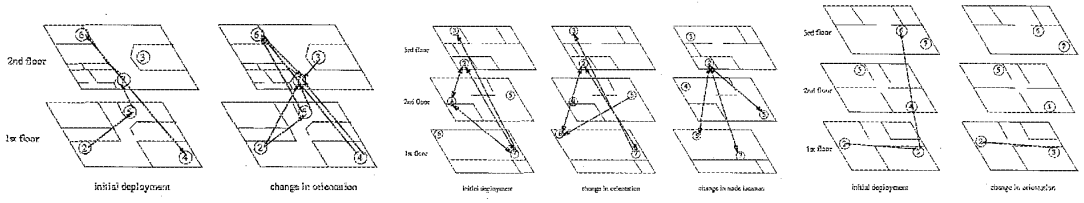


Fig. 8. Abstract home floorplans and location of links with greater than 95% loss rate at 1 mW and 11 Mbps under different configurations: (a) *ushome1* for *layout1*, *layout2*, (b) *ushome2* for *layout1*, *layout2*, and *layout3*, and (c) *ukhome1* for *layout1*, *layout2* ($* - * - link - homes$).

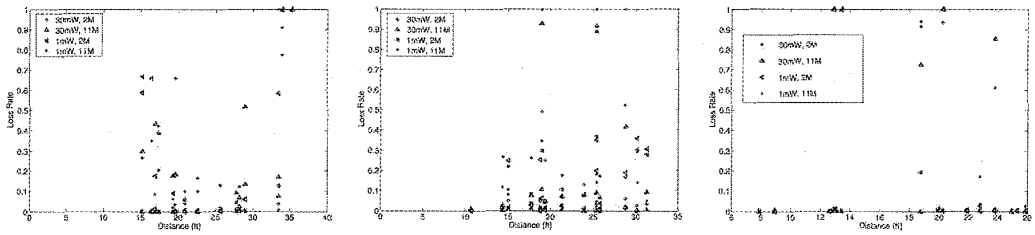


Fig. 9. Loss rate for each pair of nodes against their distance for (a) *ushome1*, (b) *ushome2*, and (c) *ukhome1* under *layout2* ($* - layout2 - link - homes$).

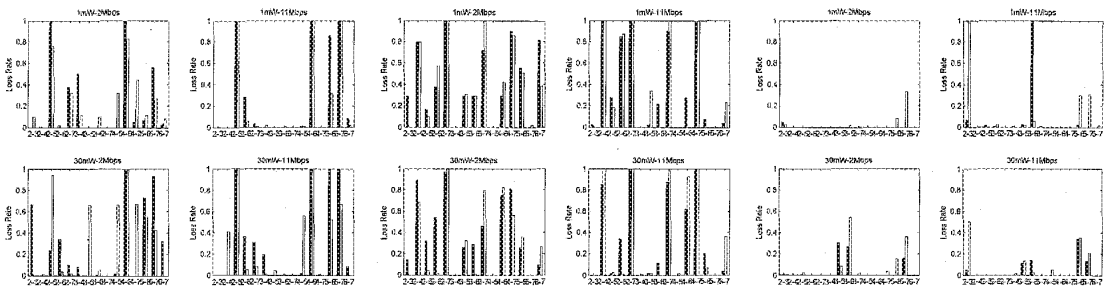


Fig. 10. Loss rate for each pair of nodes for *layout1* in (a) *ushome1*, (b) *ushome2*, and (c) *ukhome1* ($* - layout1 - link - homes$). Average loss from top-left to bottom-right: (a) 0.23, 0.26, 0.31, 0.31, (b) 0.42, 0.37, 0.35, 0.35, (c) 0.02, 0.09, 0.06, 0.07.

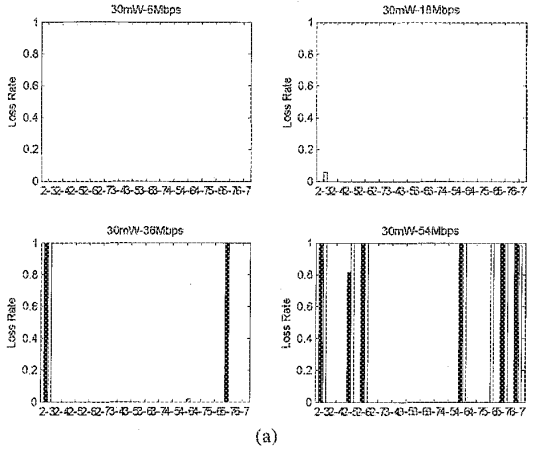
As might be expected, the characteristics of 802.11a wireless links in the home are not entirely unlike 802.11b wireless links. Packet loss rates generally increase as the link encoding rate increases. Many links are lossy, and some links are highly asymmetric. In some cases it is possible to create a nearly loss-free network at low data rates, but only at specific node locations and orientations. As with the 802.11b results, network quality appears to be sensitive to small changes in node position and orientation, as seen in *layout1* and *layout2*. Finally, we have previously confirmed that 802.11a link loss rates do not correlate with the distance between node pairs [11].

While the 802.11a results are similar to the 802.11b results, one difference is quite clear. In the home, 802.11a links appear to have a rather "binary" behavior, despite the lack of link-layer retransmissions. Link loss rates in the 802.11b experiments take on a much wider variety of values.

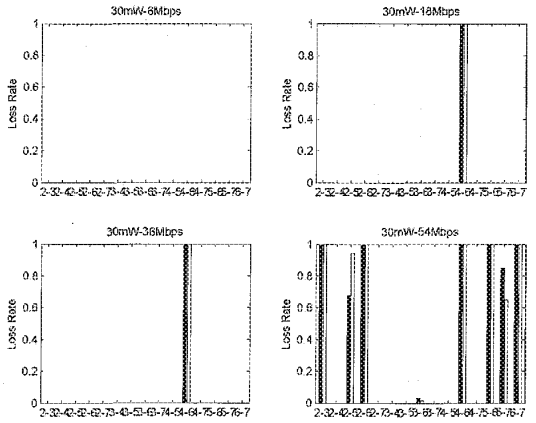
Figure 12 provides a summary comparison between 802.11a and 802.11b. In *ushome1* the 6 Mbps 802.11a links were much more reliable than either the 2 Mbps or 11 Mbps 802.11b links. Thus, one would expect 802.11a to provide better throughput in the home. However, the 54 Mbps link encoding performed very poorly between more than 40% of all node pairs. Thus, unless nodes are very optimally placed in the home, it is unlikely that 54 Mbps will be attained (a finding which is also confirmed in the next section). While one might expect the 802.11a MAC to perform better in equal environments by design, lower levels of interference from non-802.11 devices in the 5 GHz band may also contribute to the superior performance of 802.11a in the home environment.

IV. Throughput Results

In the previous section we looked into the characteristics of home wireless networks in terms of reachability. The results demonstrate that despite the small size of home networks, there may always exist nodes that cannot communicate with



(a)



(b)

Fig. 11. Loss rate for each pair of nodes for *ushome1* under IEEE 802.11a, with two different node orientations, (a) *layout1* and (b) *layout2* (*ushome1* – *layout1*/*layout2* – *link* – 11*a*). Average loss from top-left to bottom-right: (a) 0, 0, 0.1, 0.42, (b) 0, 0.06, 0.06, 0.44.

all other nodes in the network. Moreover, such communication problems cannot always be rectified by increasing the transmission power or decreasing the transmission rate (and hence employing more robust encoding).

Throughput is an additional dimension to the connectivity problem. Despite reachability, a link between two nodes may experience non-negligible loss rates, and thus suffer from intermittent connectivity and low throughput rates; a case that will be more evident for TCP traffic that is designed to react to observed losses. In this section we look at the achieved UDP and TCP throughput on our testbed networks with and without automatic link rate adaptation.

A. Fixed transmission rate using IEEE 802.11b

Figure 13(a) presents the UDP throughput rates achieved by different node pairs for different combinations of *txrate* and *txpower* in *ushome2*. We note that each node pair with a poor quality wireless link (Figure 10(b)) has zero throughput.

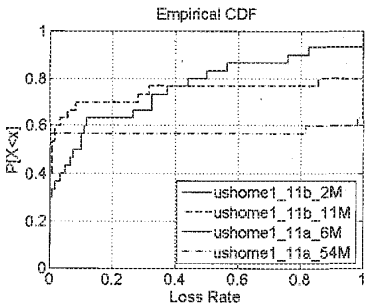


Fig. 12. Cumulative density function of loss rates under IEEE 802.11b and IEEE 802.11a in *ushome1* (*layout1*) - the 11a 6Mbps is not visible because there is no link with loss under this configuration (*ushome1* – *layout1* – *link* – 11*a*/11*b*).

An increase in transmission power, however, does not appear to improve quality in *ushome1*. Looking at the equivalent TCP throughput rates (Figure 13(b)) we notice a significant degradation in performance. This effect is expected, given that TCP is designed to react aggressively to dropped packets, lowering the overall throughput observed between two nodes with occasional packet losses.

B. Fixed transmission rate using IEEE 802.11a

In Figure 12 we conjectured that home users may not be able to make full use of the maximum transmission rate offered by IEEE 802.11a due to increased loss rates. In this subsection, we revisit this issue by measuring the throughput achieved by all node pairs in *ushome1* under the 6, 18, 36, and 54 Mbps transmission rates.

UDP throughput across transmission rates is shown in Figure 14(a)). For each node pair, as the link rate increases throughput increases, until the loss rate begins to dominate and throughput decreases. In *ushome1*, as the transmission rate increases beyond 18 Mbps, many links drop to zero throughput. Only half of the links inside *ushome1* can make use of the 54 Mbps transmission rate.

The effect of packet loss is even more dramatic on TCP (Figure 14(b)). As expected, in most cases the effect of packet loss (and perhaps increased overhead) results in lower throughput than UDP. Increasing the data rate to 54 Mbps results in extremely poor performance: no TCP connection exceeds a throughput of 10 Mbps, and half are actually zero. We note, however, that the performance observed for *txrate=18Mbps* and *txpower=30mW* is superior to the performance of 802.11b when *txrate=11Mbps* at the same *txpower* level.

C. Autorate

In our previous experiments, the link-layer transmission rate was fixed. In reality, wireless cards can adjust their encoding rate according to the quality of the wireless channel, i.e. "autorate". The autorate functionality is implemented on wireless cards such that if a high transmission rate cannot be effectively supported, the card can fall back to a lower,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1147
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

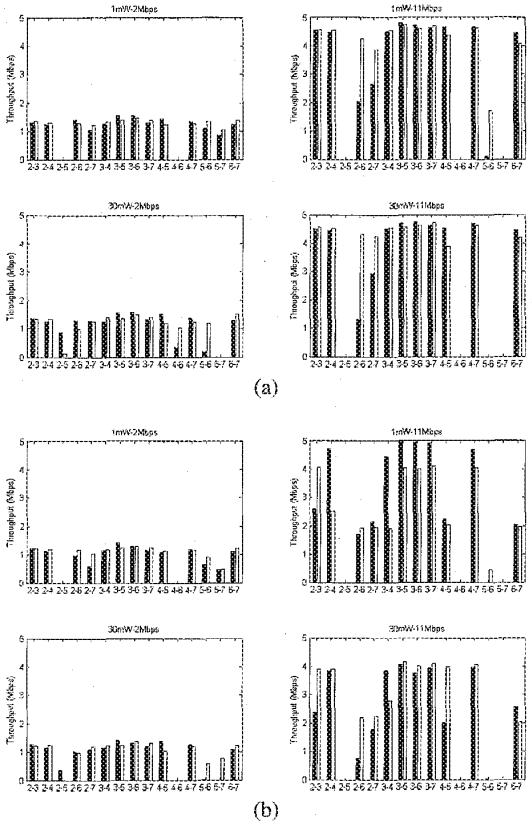


Fig. 13. (a) UDP and (b) TCP throughput rates for *ushome1* under different combinations of *txrate* and *txpower* (802.11b, $ushome1-layout1-udp/tcp-validation1$). Average throughput in Mbps from top-left to bottom-right: (a) 1.12, 3.23, 1.1, 3.14, (b) 0.93, 2.41, 0.94, 2.34.

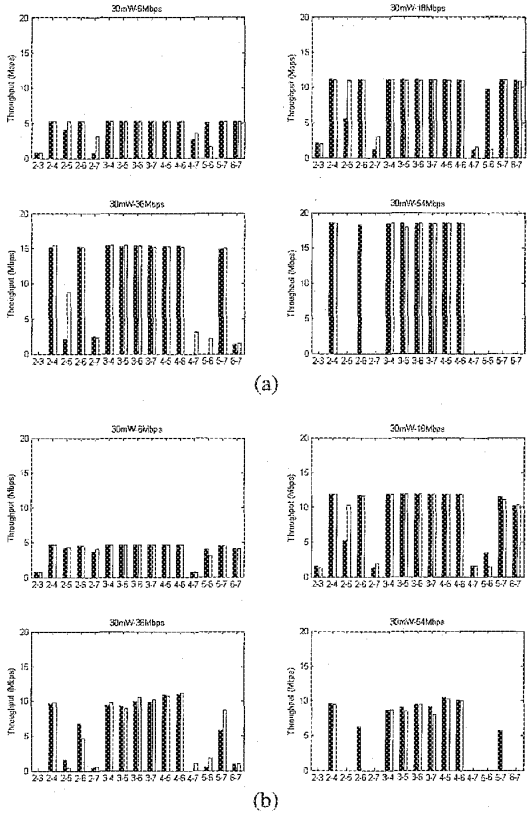


Fig. 14. (a) UDP and (b) TCP throughput rates for *ushome1* under different combinations of *txrate* and *txpower* (802.11a, $ushome1-layout1-udp/tcp-11a$). Average throughput from top-left to bottom-right: (a) 4.43, 8.64, 9.98, 9.23, (b) 3.93, 8.74, 5.85, 4.75.

more robust (in terms of encoding) transmission rate. In this section we explore the impact of autorate on our results. Unfortunately, since autorate algorithms are not standard, our results are specific to the particular implementation in our cards.

We enabled autorate on all wireless cards in our testbed in *ukhome1* and ran a series of experiments to compare UDP throughput when *txpower=30mW* with and without a fixed transmission rate, presented in Figure 15. We note that the link from node-2 to node-4 achieves zero throughput over an 11 Mbps link, while optimal throughput is achieved over a 2 Mbps link. Autorate allows nodes to adjust the link encoding rate to environmental conditions. Indeed our results demonstrate that node-2 is capable of identifying the appropriate transmission rate for the link to node-4 and using it. On the other hand, the link from node-5 to node-6 achieves a much lower throughput using autorate than when the link rate is fixed at 11 Mbps. In fact, the measured throughput over the autorate link is similar to the throughput over the 2 Mbps link. Apparently, the use of autorate in this case drops the transmission rate to 2 Mbps and does not recover. Consequently, the effective throughput achieved is under 1 Mbps.

One should bear in mind that autorate is likely to improve performance only for cases in which the minimum rate provides some connectivity. A node pair with zero throughput across all configurations will not benefit from autorate. Indeed, in Figure 15 links with zero throughput with *txrate=2Mbps* do not improve with autorate (e.g., link $5 \rightarrow 4$). On the other hand, some links with non-zero throughput at *txrate=2Mbps* and zero throughput at *txrate=11Mbps*, achieve non-zero throughput with autorate enabled.

V. Impact of Topology

The previous sections demonstrated that homes represent a challenging environment for wireless networks, particularly when used to support bandwidth-demanding applications, such as video streaming. Performance, in terms of both success rate and throughput, varies widely across links, and asymmetry is common. Performance of a given link can be difficult to predict, since the exact position and orientation of individual nodes has a greater effect than the distance between nodes.