HTTP Dynamic Streaming Specification

Version 3.0 FINAL

HTTP Dynamic Streaming Specification Version 3.0 FINAL

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 guide contains links to third-party websites that are not under the control of Adobe Systems Incorporated, and Adobe Systems Incorporated is not responsible for the content on any linked site. If you access a third-party website mentioned in this guide, then you do so at your own risk. Adobe Systems Incorporated provides these links only as a convenience, and the inclusion of the link does not imply that Adobe Systems Incorporated endorses or accepts any responsibility for the content on those third-party sites. No right, license, or interest is granted in any third party technology referenced in this guide.

Updated Information/Additional Third Party Code Information available at
http://www.adobe.com/go/thirdparty.

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

Published August 2013

Contents

1 Scope

This document describes the formats and client/server operation of the Adobe® HTTP Dynamic Streaming protocol (HDS), version 3. The intended audience of this document is implementers of HDS clients or servers.

2 Conventions

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 [RFC2119].

“MUST”, “REQUIRED” or “SHALL”, mean that the definition is an absolute requirement of the specification.

“MUST NOT” or “SHALL NOT” means that the definition is an absolute prohibition of the specification.

“SHOULD” or “RECOMMENDED” mean that there may be valid reasons to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

“SHOULD NOT” or “NOT RECOMMENDED” mean that there may be valid reasons when the particular behavior is acceptable, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

“MAY” or “OPTIONAL” mean the item is truly optional.

3 Normative References

The following documents contain provisions that, through reference in this text, constitute provisions of this specification.

[CEA708] CEA-708-D (2008), Digital Television (DTV) Closed Captioning

[F4MSPEC] Flash Media Manifest Format Specification Version 3.0

[FLVSPEC] Video File Format Specification Version 10.1
(http://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf)

[RFC2119] IETF RFC 2119, Keywords for use in RFCs to Indicate Requirements Levels, S. Bradner, March 1997, (http://www.ietf.org/rfc/rfc2119.txt?number=2119)

[RFC2142] IETF RFC 2142, URN Syntax, R. Moats, May 1997,
(http://www.ietf.org/rfc/rfc2142.txt?number=2142)

[SCTE128] ANSI/SCTE 128 (2010), AVC Video Systems and Transport Constraints for Cable Television.
(http://www.scte.org/documents/pdf/standards/ANSI_SCTE%20128%202010.pdf)

4 Terms and Definitions

Adaptive Set – a group of one or more renditions of the same content and same type that may be switched between for the purposes of adaptive bitrate switching.

Bootstrap Information – the index information used to identify which fragment and segments are available for streaming; used to "bootstrap" stream playback.

Content Stream – a media stream that typically represents a single movie, episode, or live event. Content streams often represent video-on-demand asset (VoD) or live assets.

Fragment – a small, independently decodable unit of content. In HDS, fragments are the smallest unit of media that can be addressed and delivered.

F4M – the file extension for a Flash® Media Manifest document. Flash Media Manifest files are commonly called "F4M files".

GOP – a "group of pictures"; a decodable sequence of video frames, beginning with a key frame and followed by one or more intermediate (I) frames.

HDS – Adobe HTTP Dynamic Streaming

Intermediate (I) Frame – a video frame that depends on another video frame in a GOP. This corresponds to a P-frame of B-frame in MPEG coded video.

Key frame – an independently decodable video frame. This corresponds to an IDR in MPEG coded video.

Manifest File – a file conforming to the Flash Media Manifest specification that is used to describe a presentation.

Multi-Level Manifest – manifest that is made up of several manifest file documents, which are then composed to form a single presentation. See "set-level manifest" and "stream-level manifest".

Presentation – a group of renditions that presents a content stream. A presentation contains one or more renditions.

Rendition –a single realization of a content stream intended to represent an encoding of one or more tracks of the content stream. A rendition is a logical sequence of one or more fragments.

Set-Level Manifest – a manifest document that describes only set-level details of a set of content renditions, such as bitrates, screen resolution, language, etc. A set-level manifest refers to the details of individual renditions via a URL reference.

Stream – See "content stream".

Stream-Level Manifest – manifest document that describes the details of a single content rendition (as opposed to describing the details of all renditions that make up a set).

Segment – a contiguous group of content fragments. In HDS, segments are the largest unit of content and are used to improve storage and cache efficiency.

VOD – Video on Demand. VOD content refers to static, pre-recorded streams, as opposed to live streams.

5 Introduction

5.1 Background

The HTTP Dynamic Streaming protocol was designed to allow for efficient, full-featured media delivery without the need for special purpose streaming servers. By taking advantage of the existing HTTP infrastructure, HDS can significantly reduce the cost of delivering media over the internet, while still providing access to streaming media features such as bandwidth throttling, low latency startup and seeking, and content protection.

5.2 What's New in Version 3

Version 3.0 of HDS introduces significant new changes:

- **Video-keyframe-only-renditions** are a new type of rendition that is useful for the implementation of trick modes.
- **CEA-708 Closed Captions** may be embedded within F4F fragments.
- **Adaptive Sets** are an optional construct that can be used to enhance fault tolerance by specifying backup hosts for a content stream.
- **Best effort fetch** is an optional client behavior that provides enhanced robustness in the face of server-side failures.

See Annex D. "HDS Version History (Informative)" for a full list of changes introduced in HDS 3.0.

5.3 Related Documents

[F4MSPEC] contains additional provisions that apply to the clients and servers of the HDS protocol. The reader is highly recommended to review the provisions of [F4MSPEC] in conjunction with this document.

Notable topics covered in [F4MSPEC] are:

- A specification of the *alternative content mechanism*, which enables multiple language audio tracks for a single presentation.
- A method for embedding cueing information within the manifest, which may be used to facilitate ad insertion.
- A method for embedding SMPTE timecode information within the manifest.
- The detailed semantics of manifest files.

6 System overview (Informative)

This section presents a high level overview of the logical entities involved in the HDS system. It is meant as an informative overview to aid the understanding of subsequent normative requirements.

A content stream is a media stream that represents a single movie, episode, or live event. Content streams often represent video-on-demand asset (VoD) or live assets.

A presentation is a group of renditions that present a content stream. A presentation is described by the manifest document(s). That description may be spread across multiple manifest documents, or the description may be encapsulated in a single document.

A rendition is a single realization of a content stream intended to represent an encoding of one or more tracks of the content stream. A rendition is described by the <media> element of a manifest document and is always encapsulated within a single manifest document. The description of a rendition often includes properties such as the bitrate or language code of the rendition.

A presentation typically contains more than one rendition in order to enable adaptive bitrate switching, the dynamic switching between renditions of different bitrates based on current network conditions. Another common use case for multiple renditions is to offer alternative content, such as alternative language audio for a presentation.

A rendition is logically composed of a sequence of fragments. A fragment is a named, downloadable entity that contains the audio or video data for a given interval of the presentation. For example, a fragment of a video rendition might contain the video data for a short sequence of video frames corresponding to a four second interval of time. Although audio and video fragments are the most common types of fragments, fragments may contain data other than audio or video data. Fragments are identified by a number.

The fragments of a rendition may be grouped into segments. A segment is a named group of fragments. Segments are identified by a number. The fragments of a rendition may be grouped together as a single segment, or they may be grouped into multiple segments.

A rendition is associated with a set of bootstrap information. The most notable components of the bootstrap information are the fragment run table, a table that associates stream times with fragment numbers, and the segment run table, a table that associates fragment numbers to segment numbers. The manifest specification offers multiple locations where the bootstrap information may reside, but the bootstrap information is most commonly embedded within the same manifest document as the rendition.

Multiple renditions are grouped into adaptive sets. An adaptive set is a group of renditions that may be switched between for the purpose of adaptive bitrate streaming. The adaptive sets for a presentation are described by the manifest files.

7 Fragments and Segments

7.1 Fragment Definition

Fragments are units of content that are large enough to move efficiently through the HTTP infrastructure as objects, yet small enough to be treated as discrete units for download. A fragment SHALL consist of zero or one channel of audio content, zero or one channel of video content, and zero or one channel of data content.

Each fragment SHALL be independently decodable. For fragments containing video, this means that it begins at a key frame and consists of one or more closed GOPs. This requirement ensures that seeking, adaptive bit-rate switching, etc. can always occur at a fragment boundary.

For live streams, each fragment SHOULD be very close to the same size and duration. The target size and duration of fragments SHALL be the definitions of *ideal fragment size* and *ideal fragment duration* respectively. Having fragments of a similar size and duration allows a client to utilize the minimum possible playback buffer, decreasing live lag. Fragments of identical duration also result in a more compact description in the fragment run tables, which is important for streams that expose a large DVR window. See section 8.4 “Bootstrap Information” for more details.

7.2 Segment Definition

Segments SHALL be made up of one or more fragments and used to group fragments into larger, contiguous units. The segment construct MAY be used to designate a single container within the file system or other storage area for storing multiple fragments. Using segments in this way can considerably reduce the number of files that need to be managed by the file system, and can improve HTTP cache efficiency by allowing a caching proxy to pre-fetch whole segments while processing requests for individual fragments.

7.3 Fragment Addressing

A rendition is logically a sequence of fragments. Each fragment in this sequence SHALL be identified by a sequence number that is unique to the rendition. This sequence number is known as the *fragment number.* Fragments SHALL be addressed using the fragment number assigned to that fragment. The sequence of fragment numbers for a rendition is known as a *fragment naming sequence*.

Clarified in 3.0: The fragments within a rendition SHALL be non-overlapping in time.

The first fragment number of a fragment naming sequence MAY be any non-negative integer. An HDS client SHALL NOT assume that a fragment naming sequence begins with 0.

Fragment numbers SHALL be monotonically increasing with the stream timeline. A fragment naming sequence MAY contain discontinuities. These fragment name discontinuities SHALL represent skipped numbers of the fragment naming sequence. They SHALL NOT imply skipped content but only imply skipped fragment numbers.

Proper usage of discontinuities allows servers to ensure that local variability in the structure of the content stream does not create permanent drift of the fragment naming sequence amongst multiple servers in the network serving the same HDS stream. For live use cases, this can be used to ensure that clients can easily fail over between different servers, even when the structure of the content is highly variable or individual servers are coming into and out of the cluster.

In addition to naming discontinuities, HDS also includes the concept of time discontinuities. Time discontinuities SHALL represent gaps in the stream content where content is actually missing. This type of discontinuity is commonly seen in live use cases where the encoder, ingest server, or other network element fails.

Both fragment name discontinuities and time discontinuities are advertised as part of the stream's bootstrap information. See section 8.4 "Bootstrap Information" for details.

7.4 Fragment format and URI

See Annex B. "F4F Profile" for details on the format of F4F fragments and URIs.

8 Manifest and Metadata

8.1 Manifest File

The manifest file SHALL be an XML document containing the metadata required to drive client processing of an HDS stream. The manifest file SHALL contain content metadata, fragment and segment index metadata (i.e. “bootstrap information”), and optionally content protection metadata. The manifest file SHALL be formed as specified in [F4MSPEC] and (*changed in 3.0)* SHALL indicate major version 3.

Within the manifest file, the details of each rendition SHALL be described by a<media> element. These details SHALL include the stream metadata for the individual stream, attributes such as content type (e.g. audio, video, etc.), bitrate, references to the stream’s bootstrap information, and (optional) content protection metadata.

Each manifest file SHOULD describe at least one rendition. Therefore, each manifest file SHOULD contain at least one <media> element.

The complete details of all renditions of the presentation SHALL be described either in a single manifest document or within multiple manifest documents via references. See section 8.6 “Multi-level Manifest Files” for details.

8.2 Renditions

A presentation MAY contain multiple renditions. For example, a presentation MAY contain multiple versions of audio or video content at different bitrates for use with adaptive bitrate streaming. In the manifest file, each rendition SHALL be described by a unique <media> element, as specified in [F4MSPEC].

Clarified in 3.0: All renditions of a presentation SHALL share a common presentation timeline.

8.3 Adaptive Sets

Each rendition SHALL be a member of one adaptive set. Renditions within an adaptive set MAY be switched between for the purposes of adaptive bitrate switching.

See [F4MSPEC] for the manifest file semantics of grouping renditions into adaptive sets.

8.4 Bootstrap Information

The bootstrap information for a rendition SHALL index all of the available segments and fragments in a rendition. The bootstrap information SHALL also provide a mapping from stream time to fragment number, as well as from fragment number to segment number. For a live stream, it SHALL indicate the current “live” time of the stream. Taken together, this metadata can be used by a client to “tune in” to a live stream or seek anywhere within a recorded (VOD or DVR) stream.

Each rendition MAY have a unique set of bootstrap information. When multiple renditions of a stream have an identical fragment structure, a single set of bootstrap information MAY be used to describe the renditions.

Clarified in 3.0: The bootstrap information SHOULD be nearly identical across all the renditions of an adaptive set. When possible, the bootstrap information should be identical across renditions of an adaptive set. However, in some configurations this is not achievable. In those cases, a small number of minor differences such as short-lived discontinuities, slight "live" time differences, and slight window start time differences MAY be present across renditions.

The bootstrap information SHALL be carried within the F4M manifest file's <bootstrapInfo> element, as described in [F4MSPEC]. The bootstrap information SHALL be carried in the form of an 'abst' box, as described in [FLVSPEC].

Within the 'abst' box the 'Profile', 'Live', 'MovieIdentifier', 'ServerEntryCount', 'QualityEntryCount', 'DRMData', and 'MetaData' fields SHALL be set to zero. The 'ServerEntryTable'and 'QualityEntryTable' SHALL be empty. Equivalent information is available within the F4M manifest document.

8.5 Stream Metadata

Stream metadata MAY be carried in the manifest file for each stream rendition. The stream metadata SHALL be carried in the form of an AMF "onMetadata" object, as described in [FLVSPEC]. The AMF encoded metadata SHALL be carried as BASE64 encoded binary data in the <metadata> element, a child of the rendition's <media> element, as described in [F4MSPEC].

8.6 Multi-level Manifest Files

Manifest files carry two distinct types of information. The first type of information is *set-level* metadata about all of the renditions of the presentation. Examples of set-level metadata include multiple quality levels of a video stream, or multiple language tracks for an audio stream. The second type of information is *stream-level* metadata that is specific to an individual rendition. Examples of stream-level metadata include the fragment index (bootstrap information) or DRM metadata.

When a presentation consists of multiple renditions, HDS allows multiple options for how the stream-level metadata will be distributed across manifest documents. In some cases, it might make sense to include the stream-level metadata for all renditions in a single manifest document. In other cases, it is more flexible to allow the set-level metadata to be specified in a separate manifest document from the stream-level metadata. For example, a "rendition" server might be designed to process individual renditions and be unaware of how those renditions are grouped into presentations. In this case, the "rendition" server might describe each rendition in distinct stream-level manifest documents. To provide set-level metadata, a "presentation" server such as a content management system (CMS) or web application might then have enough information to group renditions into presentations, The "presentation" server will advertise stream-level manifests that reference the renditions managed by the "rendition" server.

When defining a multi-level manifest, all <media> elements of the set SHOULD either be specified in a single manifest document, or each <media> element should have its stream-level metadata described in its own manifest document.

When a manifest document includes DVR metadata, the DVR metadata SHALL be defined in the set-level manifest document.

Details of creating a multi-level manifest document can be found in [F4MSPEC].

9 Client Operation

9.1 Client Operation for VOD streams

A client SHALL begin playback of a VOD stream by requesting a manifest file specified by an HTTP URL. If the manifest returned by the server is a set-level manifest, the client SHALL request any stream-level manifests that are appropriate for the playback session based on factors such as content type, bitrate, profile, language, etc.

For VOD streams, manifest acquisition SHOULD only take place once, at the beginning of the playback session.

For VOD streams, playback SHOULD always begin at the beginning of the first fragment specified in the bootstrap information. The decode and presentation timestamps encoded in the content MAY start at any time, including 0. A client SHOULD rebase the time such that the content appears to start at time 0.

A client SHOULD begin playback as soon as it has downloaded as many fragments as is required to fill its playback buffer. A client SHOULD NOT download additional fragments when its playback buffer is completely full. A client SHOULD resume downloading fragments when space becomes available in its playback buffer.

Since each fragment is by definition independently decodable, a client MAY treat the beginning of each fragment as a random access point. After downloading the fragment, the client MAY use the fragment random access metadata to find a random access point within the fragment, if one or more additional random access points exist.

A client SHOULD support adaptive bitrate stream switching. See section 9.4 “Adaptive Bitrate Streaming” for more information.

9.2 Client Operation for Live streams

A client SHOULD begin playback of a live stream by requesting a manifest file specified by an HTTP URL. If the manifest returned by the server is a set-level manifest, the client SHOULD immediately request any stream-level manifests that are appropriate for the playback session based on content type, bitrate, profile, language, etc.

Changed in 3.0: A client SHOULD initially download a sequence of fragments that are adjacent to the “live” end of the stream, as advertised by the initial bootstrap information. The number of fragments initially downloaded SHOULD be a number of fragments that would fill the client buffer. The client SHOULD attempt to fill its buffer using fragments advertised in the initial bootstrap information.

A client MAY be configured to begin playback further back from the “live” end of the stream. Doing so allows the client to buffer additional fragments before playback begins, which can improve stream startup time (at the cost of increasing the clients lag behind “true” live). This also has the advantage of reducing the frequency by which the client has to refresh the manifest in order to discover fragments that are more recent, since additional available fragments are listed in the current manifest.

A client SHALL continue downloading and playing recent fragments while the playback session is active. A client SHALL refresh the stream-level manifest after it has requested all of the fragments described in the manifest. A client SHOULD NOT refresh the stream-level manifest while there are still recent fragments described in the current manifest that can be requested.

For live streams, set-level manifest acquisition SHOULD only take place once, at the beginning of the playback session.

A client SHOULD support adaptive bitrate stream switching. See section 9.4 "Adaptive Bitrate Streaming" for more information.

9.3 Network Digital Video Recorder (DVR) Operation

A client MAY support network DVR capabilities when the server indicates that DVR is available on the live stream. A server SHALL indicate that DVR is available via the <dvrInfo> element of the manifest file. See section 11.6.1 "Providing Network Digital Video Recording (DVR)" for details.

When acting a DVR client, the client SHALL respect the playback boundaries expressed in the <dvrInfo> metadata, including preventing DVR playback when no <dvrInfo> element exists.

Under certain circumstances, a client might choose to present content within the fragment availability window indicated by the server when no <dvrInfo> exists, or when the content is outside the window specified by the <dvrInfo> metadata. For example, a server might provide an administrative tool that acts as an HDS client, and that client offers the ability to visually set the DVR window to be advertised. The implementation of a client that can reliably access content outside of the window specified by a <dvrInfo> element is outside the scope of this specification, and such behavior should be considered undefined.

9.4 Adaptive Bitrate Streaming

9.4.1 Initiating Adaptive Bitrate Streaming

A client SHOULD initiate adaptive bitrate streaming whenever the manifest document describes multiple available renditions of stream content.

9.4.2 Selecting an Initial Rendition

When multiple renditions of content are described in the manifest, a client SHOULD select an initial rendition for playback based on known characteristics of the playback device and past measurements of available bandwidth. For example, a client running on a high-end desktop with GPU accelerated video playback capability that typically experiences multi-megabit download rates might select a high bitrate HD video stream as its initial rendition. This client might then choose to down switch as conditions require.

9.4.3 Responding to Changes in Available Bandwidth

A client SHOULD attempt to select a rendition that best matches the available bandwidth under current network conditions. The specifics of how to achieve this are out of scope for this document, but it is recommended that each client monitor its current playback buffer and the rate at which new fragments are being downloaded to determine whether the bitrate of the current rendition is too high or too low for current conditions.

Introduced in 3.0: A client MAY only switch between renditions of the same adaptive set. For a given content and type, a client SHOULD NOT switch between renditions that are members of different adaptive sets , except during transient situations, such as while executing failover.

9.4.4 Responding to Playback Quality Issues

A client SHOULD attempt to select a rendition that best matches the media processing power of the playback device. The specifics of how to achieve this are out of scope for this document, but it is recommended that each client monitor its current playback buffer and the rate at which video frames are being dropped to determine whether the playback device is able to render the content with reasonable quality of service.

9.5 Client Buffer Length

Introduced in 3.0

A client SHOULD select a buffer length that provides minimal playback disruptions, while taking into consideration additional factors. Such factors may include (but are not limited to) expected network conditions, desired latency, desired start times, and effects on server scalability. The specifics of how to select a proper buffer length are out of scope for this document, but in the absence of other factors, it is recommended that a client select a buffer length of at least three times the ideal fragment duration.

9.6 Trick Modes

Introduced in 3.0

A client MAY use video-keyframe-only renditions to play content in trick modes , such as rewind, fast forward, fast rewind, etc. During trick play, a client SHOULD request the video-keyframe-only manifest. A client MAY select any quality video-keyframe-only rendition for playback. A client SHOULD take into consideration the available bandwidth in relation to desired playback speed when selecting between renditions. A client MAY choose to disable certain features such as adaptive bitrate switching, audio playback, seek or pause when playing in trick mode.

9.7 Failover

Changed in 3.0

A client SHOULD attempt to failover to a different server when an error condition is detected or when the client has reason to believe that a different path will demonstrate significant improvements in quality of service.

Initially, a client SHOULD select an adaptive set for playback that is most likely to result in successful playback. A client MAY assume the document ordering of adaptive sets orders the adaptive sets in reliability order. A client MAY use application specific logic or out of band information to select an adaptive set that is likely to result in successful playback. The specifics of said logic or information are out of scope for this document.

When a failover operation is initiated, a client SHOULD attempt to use the next adaptive set, as specified by the reliability ordering. If failures continue, the client SHOULD repeat the process with the subsequent adaptive set until the stream is reacquired. A client SHOULD fail stream playback after failing to acquire the stream from the last adaptive set. This recommendation is intended to avoid repeatedly cycling through the adaptive sets when all servers are unavailable, such as what can occur when a local network node goes offline.

A client SHOULD treat any 500 level response code as an error condition. A client SHOULD retry the request at least once before initiating failover.

A client MAY use application specific logic to determine what constitutes an error condition, or what action to take when an error occurs. In this case, the above recommendations do not apply. For example, a client can use application-specific logic to determine when it is appropriate to return to a primary stream source after a failure.

9.8 Best Effort Fetch

Introduced in 3.0

As described in [F4MSPEC], best effort fetch information may be provided for all renditions within a presentation or for all renditions within an adaptive set.

If best effort fetch is enabled for a rendition, a client MAY issue *best effort fetches* for that rendition, requests for fragments that are not advertised in the bootstrap information. If best effort fetch is not enabled for a rendition, a client SHALL NOT issue best effort fetches for that rendition. If the segment duration or fragment duration of the rendition is undetermined, a client SHALL NOT issue best effort fetches.

When making best effort fetches, the client SHALL assume the fragment and segment numbers of the unadvertised fragments adhere to the following formulas:

$fragmentNumber(t) = \text{floor}(t/befFragmentDuration) + 1$

if @segmentDuration is infinite (as specified by a value of “0”),

$segmentNumber(t) = 1$

otherwise

$segmentNumber(t) = \text{floor}(t/befSegmentDuration) + 1$

t, *befSegmentDuration,* and *befFragmentDuration* are expressed as a decimal number of seconds, but rounded to the nearest thousandth (millisecond).

befSegmentDuration and *befFragmentDuration* SHALL be the ideal segment duration and ideal fragment duration respectively. Their values SHALL be specified by the @segmentDuration and @fragmentDuration attributes of the <bootstrapInfo> element, as specified by [F4MSPEC]. If compatibility is desired with early revisions of the F4M 3.0 specification, clients and servers MAY implement an alternate scheme for determining *befSegmentDuration* and *befFragmentDuration*. See section 9.8.1 “Best Effort Fetch Legacy Compatibility” below.

If a best effort fetch returns a successful response, the client MAY buffer and playback the fragment returned from the best effort fetch. A client SHOULD NOT issue more than two consecutive best effort fetches unless one of the two best effort fetches succeeded or the client received a successful bootstrap update between the fetches.

9.8.1 Best Effort Fetch Legacy Compatibility

Early revisions of the HDS 3.0 and F4M 3.0 specifications used the @segmentDuration and @fragmentDuration attributes of the <bestEffortFetchInfo> element to determine the values of *befSegmentDuration* and *befFragmentDuration*. In the final version of these specifications, usage of these attributes is deprecated, as they have been superseded by the corresponding attributes of the <bootstrapInfo> element. This section outlines a scheme that clients and servers MAY implement to ensure compatibility with early versions of the specification.

If compatibility with early servers is desired, a client SHALL first check for the presence of the @segmentDuration and @fragmentDuration attributes of the <bootstrapInfo> element. If either attribute is present, a client SHALL determine the values of both *befSegmentDuration* and *befFragmentDuration* from the <bootstrapInfo> element. Otherwise (if both attributes are missing), a client SHALL determine the values of both *befSegmentDuration* and befF*ragmentDuration* from the <bestEffortFetchInfo> element.

If compatibility with early clients is desired, a server SHALL specify the @segmentDuration and @fragmentDuration attributes on both the <bootstrapInfo> element and the <bestEffortFetchInfo> element. The values and presence of these attributes in both locations SHALL be identical.

10 Server Operation

10.1 Responding to Manifest Requests

The server SHALL respond to an HTTP request for a recognized manifest file by returning a valid manifest file, as described in [F4MSPEC]. If the manifest file being requested is not known, the server SHALL respond with a standard HTTP 400 level response code.

The server SHOULD set the Content-Type header of the manifest response to “application/f4m”.

Introduced in 3.0: If it is known, the server SHOULD provide the ideal fragment duration in the manifest as described in [F4MSPEC].

10.2 Caching Manifest Files

The server SHOULD set the appropriate cache-control headers on the manifest response to control the lifetime of the manifest file within the HTTP infrastructure. For VOD content, the manifest file MAY be set as permanently cacheable. For live content, the manifest file SHOULD be set to be cached for no less than 1 second and no more than half the maximum fragment duration in the live stream. To control caching, the server SHALL set the max-age HTTP header, in compliance with the HTTP 1.1 protocol. In order to improve compatibility with non-HTTP 1.1 proxies the server SHOULD also set the Expires header, in compliance with the HTTP 1.0 protocol.

For live streams, when returning an error response for a manifest request, the server SHOULD mark the response with cache control headers in order to prevent the response from being cached. This prevents so-called *negative caching* from denying clients access to a manifest file that is only temporarily unavailable.

The server SHOULD follow standard mechanisms for facilitating HTTP 1.1 caching, including looking for the If-Modified-Since header on each HTTP request and returning an appropriate 300 level response code when the manifest content has not changed since the indicated response time.

10.3 Responding to Fragment Requests

The server SHALL respond to an HTTP request for a recognized fragment by returning a valid content fragment, as described in [FLVSPEC]. If the fragment being requested is not recognized, the server SHALL respond with a standard HTTP 400 level response code. If the server recognizes the request as being for a fragment that is currently being formed but is not yet fully available the server MAY respond with an HTTP 503 response code, indicating the client may try to request the fragment again after an appropriate back-off period. The server SHOULD set the Content-Type header of the fragment response to the mime type specified by the active HDS profile. See section 13 “HDS Profiles”.

11 HDS Versions

Introduced in 3.0

The version of the HDS protocol SHALL be defined as "<major>.<minor>", where both the major and minor fields are integers. The version of the protocol outlined by this document SHALL be version 3.0.

11.1 Compatibility

Two versions of the HDS protocol that share the same major version number SHALL be compatible. If a client or server supports protocol version *x.y*, that client or server MAY interoperate with clients or servers of version *x.z* where *z* may be any minor version number. Therefore, it follows that two versions of the protocol that only differ in the minor version of the protocol SHALL only differ in their optional or recommended practices. For example, if a client supports HDS version 2.6, it may safely interoperate with servers that support of version 2.7, provided it has adhered to the backward compatibility requirements outlined.

See [F4MSPEC] for versioning requirements specific to F4M manifests used with the F4F profile.

11.2 Guidelines for interoperating HDS 3.0 and HDS 2.0 (Informative)

Most HDS 3.0 clients can be made backward compatible with HDS 2.0 after a minimal implementation effort. That is, a client implementation that complies with HDS 3.0 typically only requires minor modifications in order to correctly interoperate with HDS 2.0 servers. The most common client change required is to allow the client to accept manifest files of version 3.0. A thorough client may also take the additional step of ignoring elements or attributes that were introduced in F4M 3.0, but this is only necessary to improve compatibility under a limit set of scenarios. When considering backward compatibility, it is important to note that an HDS 3.0 client interoperating with an HDS 2.0 server should be considered an HDS 2.0 client and therefore should not expect any HDS 3.0 features to be available.

HDS 2.0 clients are not always forward compatible with HDS 3.0 servers. That is, an HDS 2.0 client that interoperates with an HDS 3.0 server (and ignores the manifest version number) will not always behave correctly. A notable case where client behavior may be incorrect is in the handling of alternative content. If server interoperability with both HDS 2.0 clients and HDS 3.0 clients is desired, a server is recommended to offer separate F4M 2.0 and F4M 3.0 manifests.

HDS Profiles" for details.

11.3 Caching Fragments

The server SHOULD set the appropriate cache-control headers on the fragment response to control the lifetime of the fragment content within the HTTP infrastructure. For both live and VOD content, the fragment MAY be set as permanently cacheable.

For live content, the fragment MAY be set to be cached based on the duration of the DVR availability window. Fragments SHALL be cached for at least the duration of the availability window. See section 11.6 Network Digital Video Recording (DVR) for details.

To control caching, the server SHALL set the max-age HTTP header, in compliance with the HTTP 1.1 protocol. In order to improve compatibility with non-HTTP 1.1 proxies the server SHOULD also set the Expires header, in compliance with the HTTP 1.0 protocol.

Changed in 3.0: For live streams, when returning an error response for a fragment request, the server SHOULD mark the response with cache control headers of an appropriate value. The cache lifetime of error responses SHOULD be low enough to prevent the situation where clients are denied access to a fragment that is only temporarily unavailable, but also high enough to ensure that a sufficient number of requests are handled by downstream caches. In the absence of other information, a fraction of the ideal fragment duration is often a reasonable choice for the cache lifetime of error responses.

The server SHOULD follow standard mechanisms for facilitating HTTP 1.1 caching, including looking for the If-Modified-Since header on each HTTP request and returning an appropriate 300 level response code when the fragment content has not changed since the indicated response time.

11.4 Responding to Best Effort Fetch Requests

Introduced in 3.0

A server MAY signal the availability of best effort fetch via the <bestEffortFetchInfo> element of the manifest, as specified in [F4MSPEC]. If best effort fetch is enabled, fragment numbers and segment numbers SHALL conform to the formulas outlined in section 9.8 “Best Effort Fetch”. A server SHALL respond to fragment requests for fragments that are not advertised in the bootstrap. If the requested fragment is available, this response SHALL be a valid content fragment. If the fragment is not available, the server SHOULD respond with either the HTTP 503 response code, a standard HTTP 400 level response code, or a standard HTTP 500 level response code with the appropriate cache control header values.

11.5 Responding to Video-keyframe-only Fragment Requests

Introduced in 3.0

A server MAY advertise video-keyframe-only renditions. See [F4MSPEC] for the semantics of describing such renditions.

A server SHALL respond to an HTTP request for a recognized fragment of a video-keyframe-only rendition by returning a valid content fragment containing only video sequence header and first keyframe within the fragment range.

A server MAY advertise multiple video-keyframe-only renditions of the same content that differ in quality. In this case, the @bitrate attribute SHALL be specified for each video-keyframe-only rendition. The @bitrate attribute of video-keyframe-only rendition represents the bitrate of the source content from which the key frames were extracted. The semantics of this attribute are described in [F4MSPEC].

Video-keyframe-only renditions SHALL share a common presentation timeline with normal rendition. This enables a client to accurately switch between normal and video-keyframe-only renditions,

11.6 Network Digital Video Recording (DVR)

11.6.1 Providing Network Digital Video Recording (DVR)

A server MAY provide Network Digital Video Recording (DVR) capabilities. In this case, the following requirements apply:

- All fragments that are to be made available to clients for DVR SHALL be included in the stream's bootstrap information.
- The server SHALL signal to the client that DVR is supported by including the <dvrInfo> element in the manifest file. The <dvrInfo> element SHALL be formed as defined in [F4MSPEC].

The DVR window specified in the <dvrInfo> element SHALL be contained entirely within the fragment availability window defined by the stream's bootstrap information.

The beginning of the DVR window SHOULD be at least one fragment duration shorter than the fragment availability window to ensure that clients do not attempt to fetch fragments that have become unavailable as the server removes older fragments.

When providing multiple renditions of a stream for adaptive bitrate streaming, all renditions of a stream SHALL advertise the same availability window.

11.6.2 Managing the DVR Availability Window (Informative)

While under some circumstances an application might wish to expose the entire stream as part of the DVR availability window, applications will typically wish to expose only the most recent portion of the live stream. Older content is "rolled" out of the availability window, where it can be deleted or otherwise managed. To implement this behavior, a server simply removes the older entries from the stream's bootstrap information.

The first fragments advertised in the bootstrap information do not represent the beginning of the live stream. The true "start time" of the live stream can be signaled to a client via the <startTime> element, as described in [F4MSPEC].

11.7 Failover

A server SHOULD respond to any known stream error conditions with an HTTP 500 level response code. Specifically, when a server recognizes that the stream is in an invalid state it SHOULD attempt to indicate the problem to downstream caches or clients by returning a 500 level error.

Servers that are ingesting a live stream SHOULD detect when a stream is no longer active and begin responding to requests with a 500 level error. This allows downstream nodes to respond to the error condition and failover to a different node.

12 HDS Versions

Introduced in 3.0

The version of the HDS protocol SHALL be defined as "<major>.<minor>", where both the major and minor fields are integers. The version of the protocol outlined by this document SHALL be version 3.0.

12.1 Compatibility

Two versions of the HDS protocol that share the same major version number SHALL be compatible. If a client or server supports protocol version *x.y*, that client or server MAY interoperate with clients or servers of version *x.z* where *z* may be any minor version number. Therefore, it follows that two versions of the protocol that only differ in the minor version of the protocol SHALL only differ in their optional or recommended practices. For example, if a client supports HDS version 2.6, it may safely interoperate with servers that support of version 2.7, provided it has adhered to the backward compatibility requirements outlined.

See [F4MSPEC] for versioning requirements specific to F4M manifests used with the F4F profile.

12.2 Guidelines for interoperating HDS 3.0 and HDS 2.0 (Informative)

Most HDS 3.0 clients can be made backward compatible with HDS 2.0 after a minimal implementation effort. That is, a client implementation that complies with HDS 3.0 typically only requires minor modifications in order to correctly interoperate with HDS 2.0 servers. The most common client change required is to allow the client to accept manifest files of version 3.0. A thorough client may also take the additional step of ignoring elements or attributes that were introduced in F4M 3.0, but this is only necessary to improve compatibility under a limit set of scenarios. When considering backward compatibility, it is important to note that an HDS 3.0 client interoperating with an HDS 2.0 server should be considered an HDS 2.0 client and therefore should not expect any HDS 3.0 features to be available.

HDS 2.0 clients are not always forward compatible with HDS 3.0 servers. That is, an HDS 2.0 client that interoperates with an HDS 3.0 server (and ignores the manifest version number) will not always behave correctly. A notable case where client behavior may be incorrect is in the handling of alternative content. If server interoperability with both HDS 2.0 clients and HDS 3.0 clients is desired, a server is recommended to offer separate F4M 2.0 and F4M 3.0 manifests.

13 HDS Profiles

13.1 HDS Profile Model

HDS profiles are a mechanism for extending the HTTP Dynamic Streaming protocol, while ensuring interoperability between components designed to support a specific profile or set of profiles. A profile MAY be defined to support a specific new container for segments/fragments, for supporting new content protection schemes, or for any other format or behavior related change that will impact interoperability between servers, clients, or intermediary processors.

Each profile SHALL have a unique identifier assigned to it.

A profile identifier SHALL be formed as a URN as specified in [RFC2142].

In order to prevent conflicts in profile naming, a profile name SHOULD begin with an entity-specific portion, such as "profile.adobe.com".

13.2 Profile Specification within a Manifest File

Each manifest file SHOULD be marked with one or more profile identifiers. If a profile is specified, it SHALL be specified using the profile attribute of the <manifest> element. If no profile is specified, a client SHALL assume the profile is "urn://profile.adobe.com/F4F", indicating the F4F profile.

Multiple profiles MAY be specified as a list of profile identifiers, separated by spaces.

13.3 Profile Interoperability

A profile SHALL define a specific, concrete usage of the HDS specification. A server that responds to a manifest request with a manifest marked as a specific profile SHALL support all aspects of that profile. A client claiming to support a given profile SHALL support all elements of that profile.

A server MAY support multiple profiles.

A client MAY support multiple profiles.

When more than one profile is advertised a client SHOULD select the earliest profile that it recognizes and operate according to that profile. If a client does not recognize any profile advertised by the server, it SHOULD NOT attempt to play the stream.

14 Content Protection

14.1 Specifying Content Protection

A rendition MAY be protected via one or more content protection mechanisms. Content protection for a rendition SHALL be indicated via the drmAdditionalHeaderId attribute of the rendition's <media> element in the manifest file. The drmAdditionalHeaderId value SHALL match the (unique) id attribute of a single <drmAdditionalHeader> element in the manifest file. The <drmAdditionalHeader> element SHOULD contain the details of the content protection scheme or schemes used to protect the content.

A client SHOULD support playback of Flash® Access® protected content. See Annex A "Flash® Access® Content Protection for HDS (Informative)" for further details on the Flash Access content protection scheme.

14.2 Encrypting Content

The details of encrypting HDS content are specific to the content protection scheme in use. See Annex A "Flash® Access® Content Protection for HDS (Informative)" for details on encrypting F4F content for use with Flash Access.

14.3 Content Protection and Multiple Renditions

When a presentation contains multiple renditions, all renditions SHOULD use the same license and/or content encryption key. This makes switching between renditions more efficient and significantly improves the likelihood of a smooth transition. A server MAY present renditions with different licenses and/or keys when the renditions were generated at different times, by different devices, or under different administrative domains. A client SHOULD attempt to provide a smooth transition between renditions even when they are protected by different licenses and/or keys.

Annex A. Flash® Access® Content Protection for HDS (Informative)

A.1 Embedding Flash Access Metadata in the Manifest

The metadata required to acquire a Flash Access license is carried in the <drmAdditionalHeader> element of the manifest file, as described in [F4MSPEC]. Additional considerations when using license rotation may also be found in [F4MSPEC].

A.2 Encrypting Content for Flash Access

The details of encrypting HDS content for Flash Access depends on the content profile in use. See Annex B. “F4F Profile” for information on encrypting F4F content for Flash Access.

Annex B. F4F Profile

B.1 Profile Identifier

The profile identifier for F4F content SHALL be "profile.adobe.F4F". The F4F profile is the default, and SHALL be assumed when no other profile identifier is provided.

B.2 F4F Fragment Format

F4F fragments are F4V fragments, with several additional boxes that carry HDS-specific metadata. The F4F fragments SHALL be formed and identified as specified in Annex C of [FLVSPEC].

B.3 F4F Segment Format

F4F segments are complete F4V files, with several additional boxes that carry HDS-specific metadata. The F4F segments shall be formed as specified in Annex C of [FLVSPEC].

B.4 F4F Segment Indexes (F4X)

F4F segment indexes (sometimes called F4X files) are used by servers to quickly select individual fragments from within an F4F segment.

F4F segment indexes SHALL be structured as a single 'afra' box, as defined in [FLVSPEC]. The 'afra' box SHALL contain only global entries. The 'Offset' field of each entry SHALL refer to the byte offset of the first byte of the 'afra' box of a fragment within the F4F segment.

B.5 F4F Encryption for Flash Access

F4F content SHALL be encrypted in the same way as other F4V content, as specified in [FLVSPEC].

B.6 F4F Closed Captioning

Introduced in 3.0

B.6.1 Closed Captioning Format

B.6.1.0. Embedded CEA-708

An F4F fragment MAY contain embedded closed captioning data that complies with [CEA708].

If CEA-708 closed captioning is desired, content SHALL embed closed captioning data within the data track as AMF0 onCaptionInfo messages with type "708".

If CEA-708 closed captioning is desired, content that uses the H.264 codec SHALL also embed captioning data within the H.264 SEI NALU as described in [SCTE128]. Thus, closed captioning data is required to be present in both onCaptionInfo messages as well as the H.264 SEI NALUs. In order to address a special case unhandled by [SCTE128], the following additional requirements apply:

> If multiple SEI NALUs are present within a single access unit, a client SHALL render the data contained the *i*-th SEI NALU prior to rendering the the (*i*-1)-th SEI NALU.
>
> If multiple sei_messages are present within a single SEI NALU, a client SHALL render the data contained the *i*-th sei_message before the (*i*-1)-th sei_message.

B.6.1.1. onCaptionInfo message

An *onCaptionInfo* message SHALL be an AMF0 encoded data message with name "onCaptionInfo." The message SHALL have a single parameter. The type of this parameter SHALL be an AMF0 encoded object. The parameter SHALL contain the following key-value pairs:

type: SHALL be a string indicating the caption type. The value of this parameter SHALL be "708".

data: SHALL be a string that provides the caption data. The format of this string depends upon the value of "type".

B.6.1.2. onCaptionInfo.data for 708

For a type value of "708", onCaptionInfo.data SHALL be the Base64 encoding of one or more lengthPrefixedCCDataChunks, as defined below.

A *lengthPrefixedCCDataChunks* SHALL consist of a lengthPrefix followed by a ccDataChunk. A *lengthPrefix* SHALL be a 4 byte integer in network byte order expressing the length in bytes of the binary representation of the ccDataChunk that follows.

A *ccDataChunk* SHALL be the binary representation of a cc_data as specified in [CEA708], with the following two modifications:

1. The third bit (zero bit) of cc_data MAY be either 1 or 0. A value of 1 is not required to have any significance.

2. Extra, trailing bytes MAY be present beyond the final byte of the cc_data. If they are present, these bytes are not required to adhere to any format. These extra bytes, if included, SHALL be accounted for in the length prefix.

These modifications are present in order to accommodate early versions of the A/72 specification. The definitions of cc_data within the 2007 and earlier versions of this specification allowed a value of 1 for the third bit of cc_data. Within these specifications, a value of 1 indicated the presence of additional ATSC reserved data following the final byte the cc_data. In later versions of A/72, a value of 1 for the third bit of cc_data does not necessarily indicate the presence of trailing data.

When multiple lengthPrefixedCCDataChunks are present within a single onCaptionInfo message, onCaptionInfo.data SHALL be the Base64 encoding of the concatenation of each lengthPrefixedCCDataChunk binary representation.

The cc_data of the onCaptionInfo message is not required to conform to the 9600 bps rate dictated in [CEA708]. Thus, the component that generates the onCaptionInfo MAY omit padding bytes of the cc_data structure when embedding cc_datas within onCaptionInfo, provided that the resulting cc_datas' cc_count field remains accurate.

B.6.2 Closed Captioning Client Operation

An F4F client MAY support decoding and rendering of embedded closed captioning data.

B.6.2.0. Client Processing of 708

If a client that supports onCaptionInfo.data of type "708", it SHALL render onCaptionInfo messages in timestamp order. Since the timestamps of AMF0 data messages correspond to presentation time, onCaptionInfo messages SHALL follow presentation time order.

If multiple cc_data structures share the same presentation time, the cc_data structures SHALL be included within the data parameter of a single onCaptionInfo message. In this case, a client SHALL render the data contained the *i*-th lengthPrefixedCCDataChunk before the (*i*-1)-th lengthPrefixedCCDataChunk.

Annex C. F4F Profile Server Implementer's Guide (Informative)

C.1 Creating F4F Fragments

Figure C.1 shows the basic structure of an F4F fragment.

Each fragment begins with an 'afra' box. The 'afra' box both signals that this is an F4F fragment, and contains an index of the random access points within the fragment. Each fragment begins with a random access point, but additional random access points may be included in the body of the fragment.

The 'afra' box is followed by an 'abst' box that contains bootstrap information. While it is conformant to include a complete set of bootstrap information in the fragment, it isn't generally useful to do so since the complete bootstrap is typically provided within the F4M manifest. Instead, it is better to include a bootstrap update, which contains only the difference between the current bootstrap information, and the bootstrap information as it was when the previous fragment was created.

Following the 'abst' box is the standard ISO base file format 'moof' box, which designates a movie fragment. The 'moof' contains track information for all the tracks that have been multiplexed into the fragment, as well as track information for the 'rtmp' hint track.

The last box in the fragment is the 'mdat', which contains the media data for this fragment, structured as a multiplexed set of FLV tags. The media data is structured identically to a small piece of a standard FLV file, although it does not typically include an FLV header.

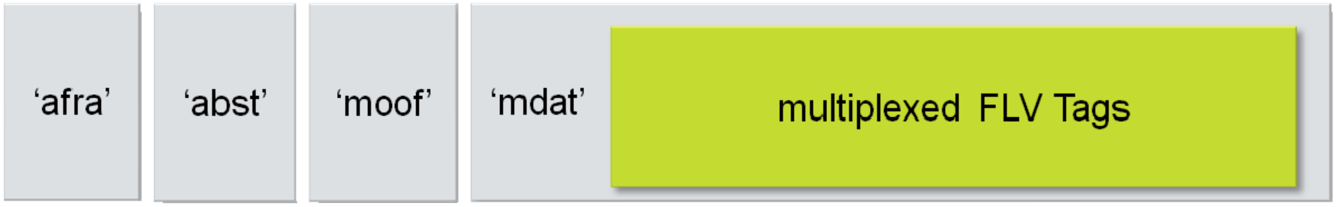

Figure C.1: F4F Fragment Structure

Figure C.2 shows a little more detail on the structure of the media data. Each element in the figure is an individual FLV tag.

The audio and video data is multiplexed in time order, just as it would be in an FLV file. The fragment will always be independently decodable, which is demonstrated here by the inclusion of a video key frame as the first video tag.

For certain audio and video codecs, such as AAC and H264 AVC, it is also necessary to include the appropriate decoder configuration record tags. These are included at the beginning of the fragment, before any of the audio or video samples. This enables a client to easily process the decoder configuration before decoding the media data. It is also recommended to repeat the decoder configuration before each video random access point within the fragment. This makes is easy for a client to quickly seek to any random access point defined in the 'abst' and begin decoding.

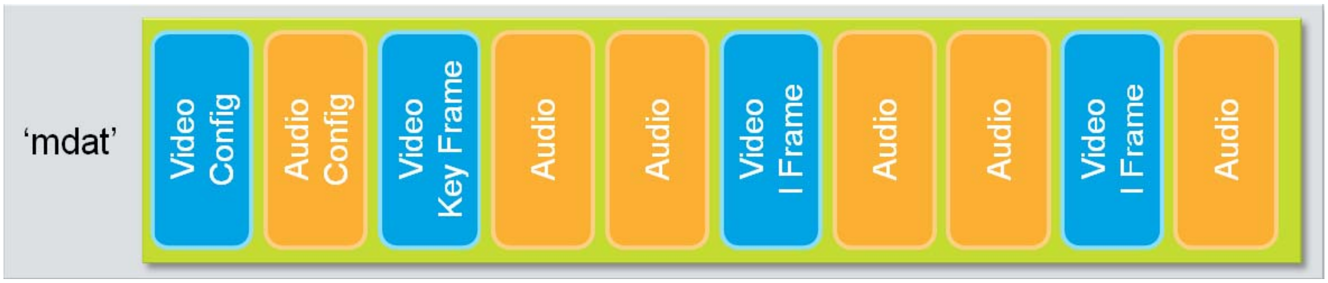

Figure C.2: Multiplexed FLV Tags within the 'rtmp' hint track

Annex D. HDS Version History (Informative)

D.1 Changes in HDS version 3.0

The following normative changes and significant informative changes were introduced in HDS version 3.0:

1. F4M Specification version 3.0 has replaced F4M version 2.0 as a normative reference. See [F4MSPEC] for a full list of normative changes introduced in F4M version 3.0.

2. The *ideal fragment size* and *ideal fragment duration* were defined and recommended server practices were introduced for these parameters. See sections 7.1 “Fragment Definition” and 10.1 “Responding to Manifest Requests”.

3. The concept of adaptive sets was defined, the optional specification of multiple adaptive sets per presentation was introduced, and a recommended usage of multiple adaptive sets for failover purposes was defined. Prior versions of HDS assumed all renditions were members of the same adaptive set, and thus implicitly grouped all renditions (minus alternative audio) into the same adaptive set. See sections 8.3 “Adaptive Sets”, 9.4.3 “Responding to Changes in Available Bandwidth”, and 9.7 “Failover”.

4. The usage of multiple <baseURL> elements for failover is no longer recommended. The usage of multiple adaptive sets is the preferred failover mechanism. See section 9.7 “Failover”.

5. Video-keyframe-only renditions were introduced along with optional practices regarding their usage for trick modes. See sections 9.6 “Trick Modes” and 11.5 “Responding to Video-keyframe-only Fragment Requests”.

6. CEA-708 Closed Captioning data may be embedded in F4F fragments. See Annex B.6 “F4F Closed Captioning”.

7. Recommended practices regarding initiating live playback were changed. See section 9.2 “Client Operation for Live streams”.

8. Recommended practices regarding client buffer length were added. See section 9.5 “Client Buffer Length”.

9. Recommended cache lifetimes for fragment error responses have been amended. See section 11.3 “Caching Fragments”.

10. An optional client-side behavior, best effort fetch, was introduced. See sections 9.8 “Best Effort Fetch” and 11.4 “Responding to Best Effort Fetch Requests”.

11. The version numbering scheme of HDS was clearly defined. Inter-version compatibility guidelines were added. See section 12 “HDS Versions”.

12. A number of requirements which were previously implied are now explicitly stated:

    a. Fragments are non-overlapping. See section 7.3 “Fragment Addressing”.

    b. Renditions within an adaptive set should have similar bootstraps. See section 8.4 “Bootstrap Information”.

    c. All renditions of a presentation share a common presentation timeline. See section 8.2 “Renditions”.

13. Ambiguous usages of the terms “rendition”, “presentation”, “stream set”, and “stream” were corrected.