SMPTE 364-2008
Revision of
SMPTE 364M-2001

SMPTE STANDARD

for Television — Declarative Data Essence — Unidirectional Hypertext Transport Protocol

Table of Contents

Approved
July 3, 2008

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in Part XIII of its Administrative Practices.

SMPTE Standard 364M was prepared by Technology Committee D27.

Intellectual Property

At the time of publication no notice had been received by SMPTE claiming patent rights essential to the implementation of this Standard. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

Introduction

The unidirectional hypertext transfer protocol, or UHTTP, is a simple, robust data transfer protocol that is designed to deliver efficiently resource data in a one-way broadcast-only environment. This transfer protocol is appropriate for delivery of HTML and other content resources using IP multicast over television vertical blanking interval (IP/VBI), in IP multicast carried in MPEG-2, or in other unidirectional transport systems. The UHTTP protocol is used to deliver television-related content resources (such as web pages, images, and scripts) which are broadcast along with a television signal.

Resources sent using the UHTTP protocol are divided into a set of data segments encapsulated in UDP packets. Typically, these packets are delivered via multicast IP, but this is not required. When delivered via IP multicast, the address and port used must be exclusively for UHTTP. That is, other UDP-based protocol data would be indistinguishable from UHTTP data, and if combined, may result in unpredictable receiver behavior. Each packet contains enough header information to begin capturing the data at any time during the broadcast, even midway through the transfer. This header contains a unique identifier (in the form of a UUID [Annex A]) that uniquely identifies the transfer, and additional information that enables the receiver to place the data following the header in the appropriate location within the transfer. Additional header information indicates to the receiver how long to continue listening for additional data.

UHTTP includes the ability to gather segments over multiple retransmissions to correct for missing packets. It is also possible to group resources together for all-or-none delivery within a UHTTP transfer. The protocol also includes a simple forward error correcting mechanism which provides for the ability to restore missing data in the event of limited packet loss.

1 Scope

This standard describes the unidirectional hypertext transfer protocol, or UHTTP. UHTTP is a one-way data transfer protocol that is designed to deliver resource data in a one-way broadcast-only environment. This transfer protocol is appropriate for delivery of HTML and other content resources using IP multicast over television vertical blanking interval (IP/VBI) and other unidirectional transport systems.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

3 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below.

IETF RFC 2616, Hypertext Transfer Protocol — HTTP/1.1

IETF RFC 2387, The MIME Multipart/Related Content-Type

ISO/IEC 11578:1996, Information Technology — Open Systems Interconnection — Remote Procedure Call (RPC), Annex A, Universal Unique Identifier

4 Data Transfer Features Enabled by the UHTTP Protocol

4.1 Robust Ddelivery: Gathering Data over MultipleTtransmissions

Data can be resent via UHTTP using the same globally unique TransferID. The data are delivered as individual segments, each of which is encapsulated within a UDP message, potentially delivered via IP multicast. Information in the header allows a receiving application to receive segments out of order or multiple

times. If the transfer data are sent repeatedly, the receiving service can fill in missing ranges using these retransmissions. This provides robust (though not necessarily reliable) data delivery. Additionally, forward error correction (FEC), using an exclusive-or-based algorithm provides for recovery of some missing segments in the face of segment loss without retransmission.

4.2 Metainformation in the Form of HTTP-Style Headers

The protocol provides for the inclusion of HTTP-style headers preceding the resource data. These headers may include information describing the content type of the resource and content location in the form of a URL (universal resource locator). It may also be used to describe groups of resources as a multipart construction. Other metainformation, including date stamping and expiration dates, may be used to provide additional information about the resource content.

4.3 UHTTP Protocol Versions

Two versions of the UHTTP protocol are defined in this standard. Version 0, the original version, limits the size of the delivered resources to approximately 4 gigabytes each. Version 1 supports delivery of resources as large as approximately 256 terabytes.

The only differences between the two versions are the value in the "version" field of the UHTTP header, the sizes of the "retransmit expiration," "resource size" and "segment start byte offset" fields of the UHTTP header, and the sizes of the "HTTP header start" and "HTTP body size" fields in the HTTPHeaderMap."

5 UHTTP Header Format

5.1 Basic UHTTP Header Format

This clause describes the format of the message packets that carry UHTTP data. It describes the information needed to create the messages using the protocol on the broadcast side and to turn those messages back into resources on the receiving side.

The UHTTP header is at the start of every UHTTP IP/UDP datagram. All values are network byte order.

For version 0 of the protocol, the UHTTP header has the format given in Figure 1. For version 1 of the protocol, the UHTTP header has the format given in Figure 2.

The fields are described in Table 1.

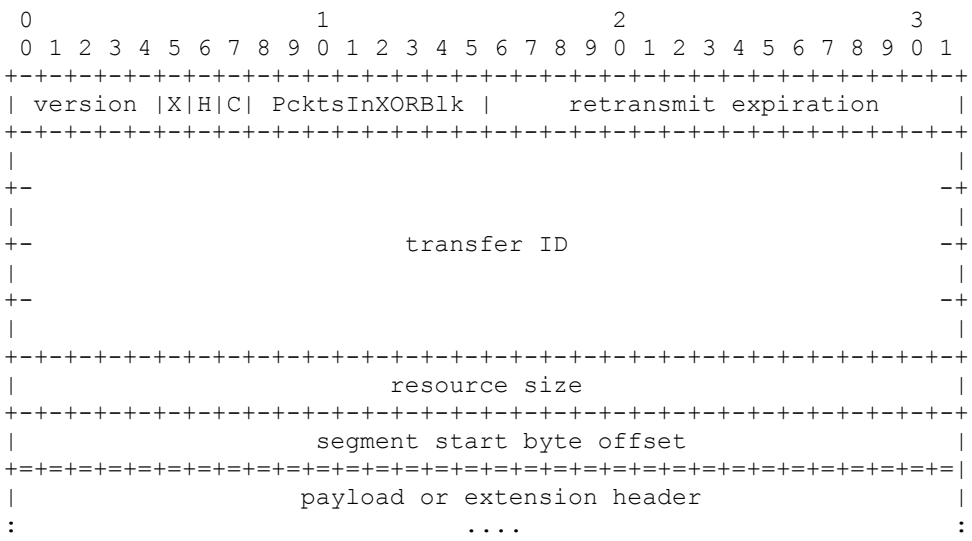

Figure 1 – UHTTP header for version 0 of the protocol

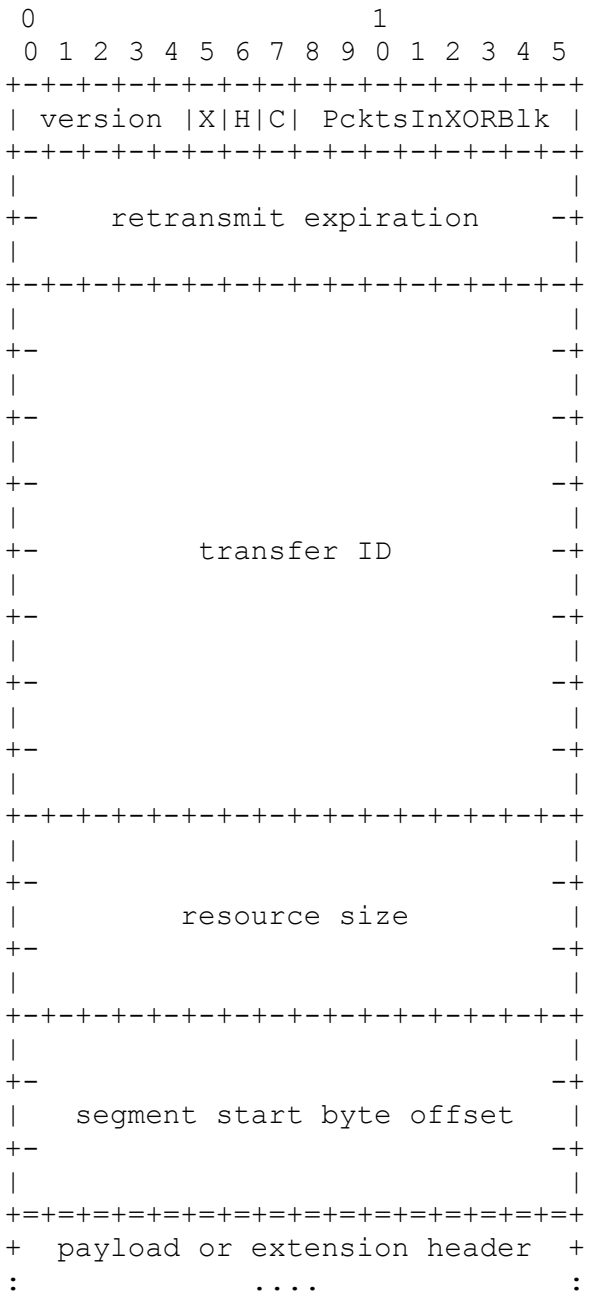

Figure 2 – UHTTP header for version 1 of the protocol

Table 1 – UHTTP fields

Name	Size	Description
Version	5 bits	Describes the version of the protocol.
ExtensionHeader (X)	1 bit	When set, this bit indicates that one or more extension header fields are present and immediately follow the main UHTTP header.
HTTPHeadersPrecede (H)	1 bit	A bit flag that, when set to 1, indicates that HTTP-style headers precede the resource data. These HTTP-style headers are considered part of the data when calculating the ResourceSize and SegStartByte fields, as well as for forward error correction. This bit must be set in all packets associated with a UHTTP transfer when HTTP-style headers precede the data. When set to zero, no HTTP-style headers precede the resource data.
CRCFollows (C)	1 bit	When the CRCFollows bit is set to 1, a 32-bit CRC is calculated and can be used to detect possible corruption in the data delivered via UHTTP. Using the MPEG-2 CRC algorithm, the CRC is calculated on the complete data, including HTTP-style headers, if any. It is then appended to the end of the data in the last logical packet. This CRC field is considered part of the data for the purposes of calculating the resource length and calculating the forward error correction. The bit must be set in all packets associated with a UHTTP transfer when a CRC is used.
PacketsInXORBlock (PcktsInXORBlk)	1 byte	Describes the number of packets in a forward error correction block, including the forward error correction packet. Set to zero when no forward error correction is used. See Section 6 for details on the forward error correction mechanism.
RetransmitExpiration	2 bytes for version 0; 4 bytes for version 1	Describes the time in seconds over which the resource may be retransmitted. This indicates how long the receiving software should wait to try to recover missing packets that follow in retransmissions of the same resource. This allows a resource to be carouseled, or sent repeatedly to increase the chances of delivery without missing segments. Set to zero if the resource will not be retransmitted. Set to maximum if the software should continue listening. The RetransmissionExpiration field should be updated to remain accurate during retransmissions, including the current transmission.
TransferID	16 bytes	Globally unique identifier (GUID [Annex A]) for the UHTTP transfer. This ID allows receiving software to identify which segments correspond to a given transfer, and determine when retransmission occurs.
ResourceSize	4 bytes for version 0; 6 bytes for version 1	Size of the complete resource data itself (excluding segment headers, XOR segments, and padding for exclusive-or correction). This length does include the length of the HTTP-style headers, if any, as well as the 4-byte CRC, if the CRCFollows bit is set to 1.
SegmentStartByteOffset (SegStartByte)	4 bytes for version 0; 6 bytes for version 1	Start offset for the first byte in the transfer for this data segment. When XOR data are used to replace missing packets, SegStartByte includes the XOR data as well as the resource data, and optional HTTP-style headers and CRC. This allows for determining where all packets fit regardless of delivery order. The exclusive-or correction packet looks like any other UHTTP packet. Its data payload is simply the exclusive-or of a number of packets that precede it in order in the data. The number of packets in an XOR block is specified in the PacketsInXORBlock field described above. The UDP packet data length for the enclosing UDP packet is used to determine the length of the segment. It is permissible to send a packet that contains UHTTP header (and optional extension headers), but without any data. If no data are included, then the SegStartByte field is ignored.

5.2 UHTTP Extension Header Format

If the ExtensionHeader flag is set in a UHTTP packet header, additional optional header fields are present. These fields appear directly after the main UHTTP header. Extension headers are optional on a packet-by-packet basis, and may appear on none, some, or all of the UHTTP packets transmitted, depending on the ExtensionHeaderType. This specification defines a single extension header type, HTTPHeaderMap (defined in section 5.2.1). It is assumed that receivers ignore any extension headers with an unknown type.

When the extension header (X) is set to a value of one, one or more extension headers may follow the UHTTP header and precede the data segment in that packet. Extension headers have the format specified in Figure 3 and Table 2.

If the ExtensionHeaderFollows bit is set, then another ExtensionHeader follows this header. If the bit is cleared, then the UHTTP data payload follows the ExtensionHeaderData (if any) immediately.

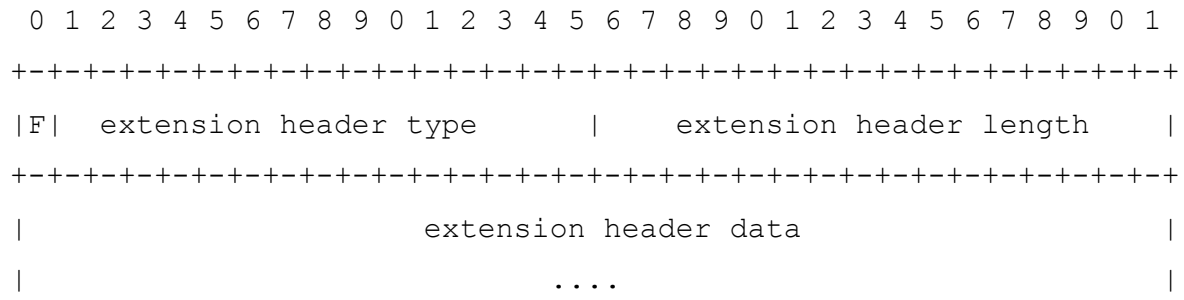

Figure 3 – Extension headers

Table 2 – Extension header formats

Name	Size	Description
ExtensionHeaderFollows (F)	1 bit	When set to 1, this field indicates that another extension header follows this one. When set to 0, the UHTTP data payload follows this extension header.
ExtensionHeaderType	15 bits	Identifies the extension header type. (1 = HTTPHeaderMap; all other values reserved.)
ExtensionHeaderDataSize	2 bytes	Describes the length of the complete extension header data in bytes. Zero indicates that there is no ExtrensionHeaderData following.
ExtensionHeaderData		The variable length data for this extension header. The length of the ExtensionHeaderData field is indicated by the ExtensionHeaderDataSize.

5.2.1 HTTPHeaderMap extension header

One ExtensionHeaderType is defined for this specification. When ExtensionHeaderType is set to a value of 1, then the ExtensionHeaderData field contains an HTTPHeaderMap. An HTTPHeaderMap extension header may optionally be included whenever the UHTTP transfer contains HTTP-style header information (as indicated by the HTTPHeadersPrecede bit in the main UHTTP header). If HTTPHeaderMap extension headers are used, they should be included in every packet in a UHTTP transfer that contains header, body, or forward error correction (FEC) data.

The HTTPHeaderMap consists of one or more sets of the fields described in Figures 4 and 5 and Table 3.

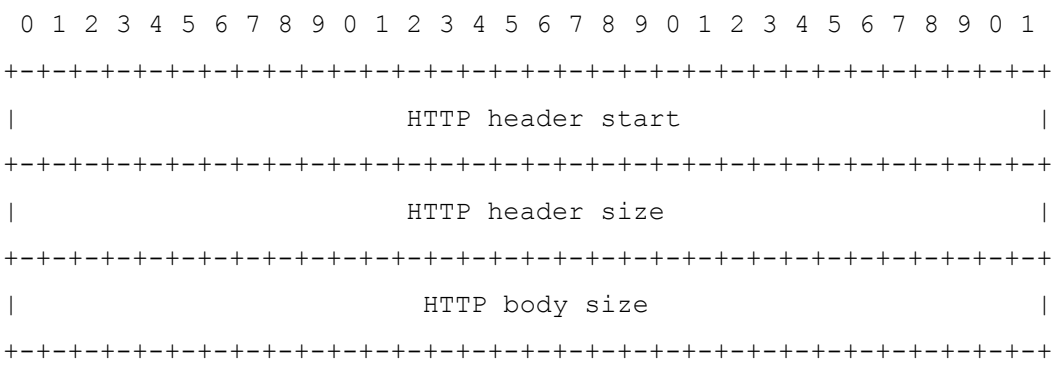

Figure 4 – HTTPHeaderMap for version 0 of the protocol

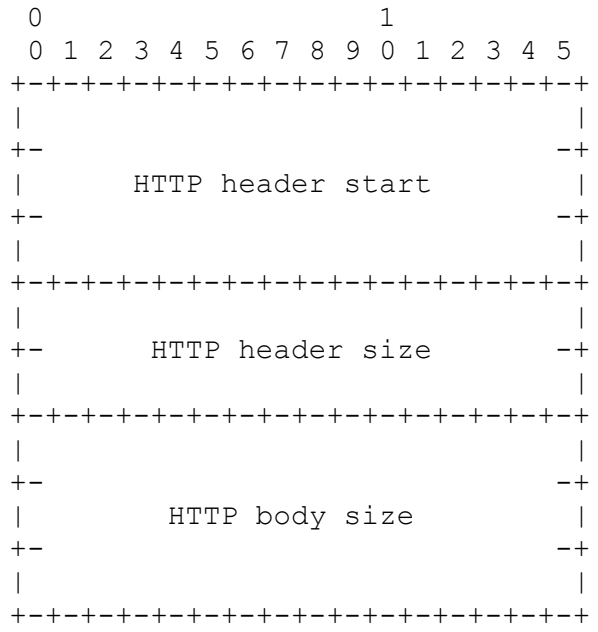

Figure 5 – HTTPHeaderMap for version 1 of the protocol

Table 3 – HTTPHeaderMAP fields

Name	Size	Description
HTTPHeaderStart	4 bytes for version 0; 6 bytes for version 1	This field indicates an offset into the UHTTP data, in bytes, where an HTTP-style header is found. The offset is calculated from the beginning of the corrected UHTTP data, and does not include the FEC data when the FEC mechanism is used.
HTTPHeaderSize	4 bytes	This field indicates the length of the HTTP-style header, in bytes, including the HTTP-style header fields, the terminating pair of carriage-return/newline character sequences, and any preceding multipart boundary lines.
HTTPBodySize	4 bytes for version 0; 6 bytes for version 1	This field indicates the length of the data body, in bytes, associated with the HTTP header described in this map entry.

When the UHTTP transfer consists of a single (i.e., nonmultipart) resource, the HTTPHeaderMap contains a single set of HTTPHeaderMap fields, of total size 12 bytes for version 0 of the protocol and total size 16 bytes for version 1 of the protocol. The HTTPHeaderStart, in this case, will be set to zero and the HTTPHeaderSize will be set to the sum of the length of the HTTP-style header fields and all separating carriage-return/newline characters. The HTTPBodySize field will contain the size, in bytes, of the body data related to that header field.

When a UHTTP transfer contains multiple resources (as specified by a multipart content-type), multiple sets of HTTPHeaderMap groups may be included in the HTTPHeaderMap data, each indicating the offset and size of the HTTP-style headers for each resource, (including any multipart boundary lines, HTTP-style header fields, and separating carriage-retum/newline characters), as well as the size of the body relating to each header.

When including HTTPHeaderMap data, senders must, at a minimum, include HTTPHeaderMap entries for each HTTP-style header that is partially or completely included in a given packet. Additionally, when forward error correction is used in UHTTP transfers that contain HTTPHeaderMaps extension headers, senders must include HTTPHeaderMap entries as extension headers in FEC-data packets for all HTTP-style header sections that may be corrected by the FEC packet. Senders are free to include additional HTTPHeaderMap entries in any packet beyond the minimum.

6 Forward Error Correction Mechanism

In addition to the ability to retransmit data and have missing segments filled in during retransmissions, this transfer protocol can also include extra data packets that can be used for simple missing packet error correction. When the PacketsInXORBlock header field is set to zero, there is no exclusive-or forward error correction. When nonzero, forward error correction packets may be sent to allow for correction of missing or corrupted packets by the receiver, and all segments must be of the same length. It is permissible to send packets with no data payload (but with UHTTP headers and optional extension headers). In this case, the packet is ignored in the calculation of forward error correction.

In blocks of PacketsInXORBlock equal-size data segments, the last data segment in the block contains the exclusive-or of the preceding segments (PacketsInXORBlock-1). Each byte of the data in this XOR segment is the exclusive-or of the corresponding byte in each of the other segments in that data block. If the data are thought of as laid out separated into consecutive segments, then after every PacketsInXORBlock-1 segment, another segment is inserted that looks exactly like resource data and has its own position offset into the transfer like resource data. The data in that segment are the exclusive-or of the previous packets in that block. XOR data in the XOR packet are the exclusive-or data segment contents only, including the HTTP-style header fields but not including the UHTTP header that is also in the packet.

If forward error correction is used, the data payload of all packets must be the same size. The packet containing the data at the end of the file (including the optional CRC) must be zero filled. Packets between this packet and the last XOR packet need not be sent since it is assumed that the receiver knows their contents are all zeros, as it was provided the overall length in the UHTTP resource size field. If they are sent, they must be zero filled after the segment header; if not sent, they are assumed to contain a payload of all zero bytes for the purposes of forward error correction calculations. The last XOR packet has the value SegStartByte calculated to be just as if zero-filled extra packets were sent, but there is no requirement to send those empty packets.

To correct for a single missing packet in a block, it is assumed that the receiver calculates the exclusive-or the data payload of the packets that arrived with the XOR data segment for that block. An important point is that segments can be sent in any order since each segment including the XOR segment indicates where in order they belong. By sending segments (including the XOR packets) out of order, there may be protection against burst errors that lose successive packets. When retransmitting a UHTTP transfer, a different order of segments can be used in each retransmission to avoid different types of burst errors. This protocol allows the headend (broadcast side) tools to decide how to order sending packets, thereby providing a great deal of

flexibility. The receiving side does not need to know the transmission order (consistent with the fact that in general it cannot know the transmission order for IP multicast delivery).

7 HTTP-Style Headers used in UHTTP

The UHTTP transfer protocol can be used to deliver resources via a broadcast medium, which can simultaneously deliver resources, including web-related content, to large numbers of users simultaneously. HTTP-style header fields can be included in the UHTTP data to provide metainformation about the content, including identifying URIs (universal resource identifiers), expiration dates, and content type and encoding. The use of HTTP-style headers is optional in UHTTP, but is required for resources intended to be interpreted as web content.

Note: Although HTTP headers preceding resource content are optional in the UHTTP protocol, they are required when the protocol is used for delivery of declarative data essence content.

HTTP-style headers, as specified by HTTP/1.1 (IETF RFC 2616), precede the resource content just as in HTTP transfers when resources are sent as a response to a HTTP GET or POST command. The HTTP-style headers may provide additional information to the browser; for example, the expiration time for the resource. The HTTP-style headers precede the body of the resource data, and are treated as part of the data payload. The protocol header and its version imply the equivalent HTTP response line (e.g., HTTP/1.1 200 OK). The header fields that are expected to be supported by all receivers are listed below and should be interpreted per the HTTP/1.1 (IETF RFC 2616) specification. No assumption should be made for support of other header fields.

7.1 Supported HTTP-Style Headers

Header fields required for every resource when using HTTP style headers:

- Content-Length:
- Content-Location:

Recommended HTTP-style header fields:

- Content-Type:

Optional HTTP-style header fields:

- Content-Base:
- Content-Encoding:
- Content-Language:
- Content-Style-Type:
- Date:
- Expires:
- Last-Modified:

UHTTP transfers that utilize HTTP-style headers shall contain the required headers listed above (Content-location and Content-length), to ensure that an appropriate URI is specified for the resource and to ensure that the resource data are delivered in whole. No other header fields are strictly required, but may provide useful information to the receiver in determining the disposition of the content.

Note: It is assumed that receivers compliant with the declarative data essence content and IP multicast encapsulation specifications support gzip (IETF RFC 1952) content encoding, as specified by the Content-Encoding HTTP header field.

It is assumed that receivers will decode the headers and data and store them in a local cache system, and that different receiver platforms will have different cache sizes for storing local resources, which may or may

not correspond to traditional web browser caches. Resources transmitted with Content-location header fields, which contain http: URLs, identify resources, which are also available via an HTTP request specified by that URL. The use of Content-Location: headers with local identifier, or lid: URLs is intended to mirror resource delivery to a local cache without requiring that the data be available on the Internet.

It is assumed that receiving platforms should take into consideration that the same resources will likely be sent repeatedly to provide resources for users who tune in late. HTTP-style header fields can be examined by receivers to determine if the resource is already present, and so can be ignored. The Date:, Expires:, and Last-Modified: headers can be used by receivers to determine the lifetime of a resource in a given receiver cache.

8 Packaging More than One Resource

It is possible to send multiple resources in a single UHTTP transfer by packaging them into a single multi-part resource for all-or-nothing transfer. In this case, the HTTP-style Content-Type: header field would have a value of multipart/related (IETF RFC 2387). When this type is used, the HTTP-style header is ended as usual and is followed by the usual boundary structure for multipart/related separating multiple related resources that each use the HTTP-style header formats. This is a mechanism to package multiple related resources together in a single all-or-nothing transfer. The HTTP-style headers for individual sub-parts describe only that subpart, and are interpreted as per the HTTP/1.1 specification (IETF RFC 2616). In this case, it may be convenient to specify a Content-Base: for the entire package and then specify relative URLs for each of the Content-Location: headers for subsequent subparts. The multipart/related content type should be used as per IETF RFC 2387.

An example using HTTP scheme URLs:

Content-Base: http://www.blahblah.com/

Content-Length: 3495

Content-Type: Multipart/Related; boundary=example98203804805

–example98203804805

Content-Location: resource1.html

Content-Length: 495

Content-Type: text/html

... Resource data for resource1.html ...

... <IMG src=image.jpg ...

–example98203804805

Content-Location: image1.jpg

Content-Length: 1495

Content-Type: image/jpeg

... Resource data for image1.jpg ...

–example98203804805

9 Security Considerations

There are a number of security issues associated with the use of UHTTP to deliver content on public broadcast networks, many of which are shared with electronic mail systems. When HTTP-style headers are used to identify the resources in a UHTTP transfer, it is possible for the sender to misrepresent the content with URIs which do not match the transmitted content. The security issues associated with this mislabeling, as well as the security issues associated with the use of HTML content which is broadcast are the same as those identified in clause 11.1 of IETF RFC 2387. Additionally, there are security issues associated with the caching of data transmitted via UHTTP which are the same as those identified in clause 11.2 of IETF RFC 2387.

It should be noted that many broadcast systems employ conditional access systems (satellite and cable television, for example), which provide a level of security for the broadcast channel leading up to the receiver. Unfortunately, it may be possible to insert UHTTP data earlier in the broadcast chain at points which are less secure (on video tape to be played into the system, for example), so applications using UHTTP which cannot ensure end-to-end security of the broadcast system should employ additional security mechanisms.

Annex A (Informative)
UUID (Universally Unique Identifier)

Definition: UUID (universally unique identifier), also known as GUID (globally unique identifier), is a fixed-size, 128-bit identifier that is unique across both space and time, with respect to the space of all UUIDS.

The generation of UUIDs does not require that a registration authority be contacted for each identifier. Instead, it requires a unique value over space for each UUID generator. This spatially unique value is specified as an IEEE 802 address, which is usually already available to network-connected systems. This 48-bit address can be assigned based on an address block obtained through the IEEE registration authority.

When IEEE 802 address is available to a system desiring to generate a UUID, the method described in ISO/IEC 11578, annex A (UUID), should be used. If no IEEE 802 address is available, other methods that provide a way to generate a probabilistically unique one that cannot conflict with any properly assigned IEEE 802 address, may be used. SMPTE 330M (UMID), for example, describes a method similar to the ISO method, but more suited to television systems where some of the ISO fields are not easily created.

Annex B (Informative)
Bibliography

ANSI/IEEE 802-2001, Standard for LAN/MAN (Local Area Network/ Metropolitan Area Network): Overview and Architecture

SMPTE 330M-2004, Television — Unique Material Identifier (UMID)

IETF RFC 1952, GZIP File Format Specification Version 4.3

Revision Notes

This version incorporates Amendment #1 to SMPTE 364M approved January 25, 2008. The purpose of this revision is to add a new header version that supports >4GB file sizes. The changes are summarized below.

1. The Introduction (previously Section 3) now follows the Foreword.

2. "Section 2, Conformance Notation" has been added.

3. The Normative References section has been renumbered Section 3.

4. "Section 4.3, UHTTP Protocol Versions" has been added.

5. Section 5, UHTTP Header Format has been revised.