Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2018

J. Yasskin
Google

June 13, 2018

Bundled HTTP Exchanges
draft-yasskin-wpack-bundled-exchanges-00

Abstract

Bundled exchanges provide a way to bundle up groups of HTTP request+response pairs to transmit or store them together. They can include multiple top-level resources with one identified as the default by a manifest, provide random access to their component exchanges, and efficiently store 8-bit resources.

Note to Readers

Discussion of this draft takes place on the ART area mailing list (art@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=art [1].

The source code and issues list for this draft can be found in https://github.com/WICG/webpackage [2].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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 https://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 December 15, 2018.

Table of Contents

1. Introduction

To satisfy the use cases in [I-D.yasskin-webpackage-use-cases], this document proposes a new bundling format to group HTTP resources. Several of the use cases require the resources to be signed: that's provided by bundling signed exchanges ([I-D.yasskin-http-origin-signed-responses]) rather than natively in this format.

1.1. Terminology

Exchange (noun) An HTTP request+response pair. This can either be a request from a client and the matching response from a server or the request in a PUSH_PROMISE and its matching response stream. Defined by Section 8 of [RFC7540].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Mode of specification

This specification defines how conformant bundle parsers work. It does not constrain how encoders produce a bundle: although there are some guidelines in Section 4, encoders MAY produce any sequence of bytes that a conformant parser would parse into the intended semantics.

This specification uses the conventions and terminology defined in the Infra Standard ([INFRA]).

2. Semantics

A bundle is logically a set of HTTP exchanges, with a URL identifying the manifest(s) of the bundle itself.

While the order of the exchanges is not semantically meaningful, it can significantly affect performance when the bundle is loaded from a network stream.

A bundle is parsed from a stream of bytes, which is assumed to have the attributes and operations described in Section 2.1.

Bundle parsers support two operations, Load a bundle's metadata (Section 2.2) and Load a response from a bundle (Section 2.3) each of which can return an error instead of their normal result.

A client is expected to load the metadata for a bundle as soon as it start downloading it or otherwise discovers it. Then, when fetching ([FETCH]) a request, the cliend is expected to match it against the requests in the metadata, and if one matches, load that request's response.

2.1. Stream attributes and operations

- A sequence of *available bytes*. As the stream delivers bytes, these are appended to the available bytes.
- An *EOF* flag that's true if the available bytes include the entire stream.
- A *current offset* within the available bytes.
- A *seek to offset N* operation to set the current offset to N bytes past the beginning of the available bytes. A seek past the end of the available bytes blocks until N bytes are available. If the stream ends before enough bytes are received, either due to a network error or because the stream has a finite length, the seek fails.
- A *read N bytes* operation, which blocks until N bytes are available past the current offset, and then returns them and seeks forward by N bytes. If the stream ends before enough bytes are received, either due to a network error or because the stream has a finite length, the read operation returns an error instead.

2.2. Load a bundle's metadata

This takes the bundle's stream and returns a map ([INFRA]) of metadata containing at least keys named:

requests A map ([INFRA]) whose keys are [FETCH] requests for the HTTP exchanges in the bundle, and whose values are opaque metadata that Load a response from a bundle can use to find the matching response.

manifest The URL of the bundle's manifest(s). This is a URL to support bundles with multiple different manifests, where the client uses content negotiation to select the most appropriate one.

The map may include other items added by sections defined in the Web Bundle Section Name Registry.

This operation only waits for a prefix of the stream that, if the bundle is encoded with the "responses" section last, ends before the first response.

This operation's implementation is in Section 3.2.

2.2.1. Load a bundle's metadata from the end

If a bundle's bytes are embedded in a longer sequence rather than being streamed, a parser can also load them starting from a pointer to the last byte of the bundle. This returns the same data as Section 2.2.

This operation's implementation is in Section 3.2.5.

2.3. Load a response from a bundle

This takes the sequence of bytes representing the bundle and one request returned from Section 2.2 with its metadata, and returns the response ([FETCH]) matching that request.

This operation can be completed without inspecting bytes other than those that make up the loaded response, although higher-level operations like proving that an exchange is correctly signed ([I-D.yasskin-http-origin-signed-responses]) may need to load other responses.

Note that this operation uses the metadata for a particular request returned by Section 2.2, while a client will generally want to load the response for a request that the client generated. TODO: Specify how a client determines the best available bundled response, if any, for that client-generated request, in this or another document.

This operation's implementation is in Section 3.3.

3. Format

3.1. Top-level structure

This section is non-normative.

A bundle holds a series of named sections. The beginning of the bundle maps section names to the range of bytes holding that section. The most important section is the "index" (Section 3.2.1), which similarly maps serialized HTTP requests to the range of bytes holding that request's serialized response. Byte ranges are represented using an offset from some point in the bundle _after_ the encoding of

the range itself, to reduce the amount of work needed to use the shortest possible encoding of the range.

Future specifications can define new sections with extra data, and if necessary, these sections can be marked "critical" (Section 3.2.3) to prevent older parsers from using the rest of the bundle incorrectly.

The bundle is roughly a CBOR item ([I-D.ietf-cbor-7049bis]) with the following CDDL ([I-D.ietf-cbor-cddl]) schema, but bundle parsers are required to successfully parse some byte strings that aren't valid CBOR. For example, sections might have padding between them, or even overlap, as long as the embedded relative offsets cause the parsing algorithms in this specification to return data.

```
webbundle = [
  ; &#127760;&#128230; in UTF-8.
  magic: h'F0 9F 8C 90 F0 9F 93 A6',
  section-offsets: bytes .cbor {* ($section-name .within tstr) =>
                                  [ offset: uint, length: uint] },
  sections: [* $section ],
  length: bytes .size 8,  ; Big-endian number of bytes in the bundle.
]

$section-name /= "index" / "manifest" / "critical" / "responses"

$section /= index / manifest / critical / responses

responses = [*response]
```

3.2. Load a bundle's metadata

A bundle holds a series of sections, which can be accessed randomly using the information in the "section-offset" CBOR item:

```
section-offsets = {* tstr => [ offset: uint, length: uint] },
```

Offsets in this item are relative to the _end_ of the section-offset item.

To implement Section 2.2, the parser MUST run the following steps, taking the "stream" as input.

1. Seek to offset 0 in "stream". Assert: this operation doesn't fail.

2. If reading 10 bytes from "stream" returns an error or doesn't return the bytes with hex encoding "84 48 F0 9F 8C 90 F0 9F 93

A6" (the CBOR encoding of the 4-item array initial byte and 8-byte bytestring initial byte, followed by 🌐📦 in UTF-8), return an error.

3. Let "sectionOffsetsLength" be the result of getting the length of the CBOR bytestring header from "stream" (Section 3.4.2). If this is an error, return that error.

4. If "sectionOffsetsLength" is TBD or greater, return an error.

5. Let "sectionOffsetsBytes" be the result of reading "sectionOffsetsLength" bytes from "stream". If "sectionOffsetsBytes" is an error, return that error.

6. Let "sectionOffsets" be the result of parsing one CBOR item (Section 3.4) from "sectionOffsetsBytes", matching the section-offsets rule in the CDDL ([I-D.ietf-cbor-cddl]) above. If "sectionOffsets" is an error, return an error.

7. Let "sectionsStart" be the current offset within "stream". For example, if "sectionOffsetsLength" were 52, "sectionsStart" would be 64.

8. Let "knownSections" be the subset of the Section 6.2 that this client has implemented.

9. Let "ignoredSections" be an empty set.

10. For each ""name"" key in "sectionOffsets", if ""name""'s specification in "knownSections" says not to process other sections, add those sections' names to "ignoredSections".

11. Let "metadata" be an empty map ([INFRA]).

12. For each ""name""/["offset", "length"] triple in "sectionOffsets":

    1. If ""name"" isn't in "knownSections", continue to the next triple.

    2. If ""name""'s Metadata field is "No", continue to the next triple.

    3. If ""name"" is in "ignoredSections", continue to the next triple.

    4. Seek to offset "sectionsStart + offset" in "stream". If this fails, return an error.

5. Let "sectionContents" be the result of reading "length" bytes from "stream". If "sectionContents" is an error, return that error.

6. Follow ""name""'s specification from "knownSections" to process the section, passing "sectionContents", "stream", "sectionOffsets", "sectionsStart", and "metadata". If this returns an error, return it.

13. If "metadata" doesn't have entries with keys "requests" and "manifest", return an error.

14. Return "metadata".

3.2.1. Parsing the index section

The "index" section defines the set of HTTP requests in the bundle and identifies their locations in the "responses" section.

```
index = {* headers => [ offset: uint,
                        length: uint] }
```

To parse the index section, given its "sectionContents", the "sectionsStart" offset, the "sectionOffsets" CBOR item, and the "metadata" map to fill in, the parser MUST do the following:

1. Let "index" be the result of parsing "sectionContents" as a CBOR item matching the "index" rule in the above CDDL (Section 3.4). If "index" is an error, return an error.

2. Let "requests" be an initially-empty map ([INFRA]) from HTTP requests ([FETCH]) to structs ([INFRA]) with items named "offset" and "length".

3. For each "cbor-http-request"/["offset", "length"] triple in "index":

    1. Let "headers"/"pseudos" be the result of converting "cbor-http-request" to a header list and pseudoheaders using the algorithm in Section 3.5. If this returns an error, return that error.

    2. If "pseudos" does not have keys named ':method' and ':url', or its size isn't 2, return an error.

    3. If "pseudos[':method']" is not 'GET', return an error.

Note: This could probably support any cacheable (Section 4.2.3) of [RFC7231]) and safe (Section 4.2.1 of [RFC7231]) method, matching PUSH_PROMISE (Section 8.2 of [RFC7540]), but today that's only HEAD and GET, and HEAD can be served as a transformation of GET, so this version of the specification keeps the method simple.

4. Let "parsedUrl" be the result of parsing ([URL]) "pseudos[':url']" with no base URL.

5. If "parsedUrl" is a failure, its fragment is not null, or it includes credentials, return an error.

6. Let "http-request" be a new request ([FETCH]) whose:

   + method is "pseudos[':method']",

   + url is "parsedUrl",

   + header list is "headers", and

   + client is null.

7. Let "streamOffset" be "sectionsStart + section-offsets["responses"].offset + offset". That is, offsets in the index are relative to the start of the "responses" section.

8. If "offset + length" is greater than "sectionOffsets["responses"].length", return an error.

9. Set "requests"["http-request"] to a struct whose "offset" item is "streamOffset" and whose "length" item is "length".

4. Set "metadata["requests"]" to "requests".

3.2.2. Parsing the manifest section

The "manifest" section records a single URL identifying the manifest of the bundle. The bundle can contain multiple resources at this URL, and the client is expected to content-negotiate for the best one. For example, a client might select the one with an "accept" header of "application/manifest+json" ([appmanifest]) and an "accept-language" header of "es-419".

manifest = text

To parse the manifest section, given its "sectionContents" and the "metadata" map to fill in, the parser MUST do the following:

1. Let "urlString" be the result of parsing "sectionContents" as a CBOR item matching the above "manifest" rule (Section 3.4. If "urlString" is an error, return that error.

2. Let "url" be the result of parsing ([URL]) "urlString" with no base URL.

3. If "url" is a failure, its fragment is not null, or it includes credentials, return an error.

4. Set "metadata["manifest"]" to "url".

3.2.3. Parsing the critical section

The "critical" section lists sections of the bundle that the client needs to understand in order to load the bundle correctly. Other sections are assumed to be optional.

```
critical = [*tstr]
```

To parse the critical section, given its "sectionContents" and the "metadata" map to fill in, the parser MUST do the following:

1. Let "critical" be the result of parsing "sectionContents" as a CBOR item matching the above "critical" rule (Section 3.4). If "critical" is an error, return that error.

2. For each value "sectionName" in the "critical" list, if the client has not implemented sections named "sectionName", return an error.

This section does not modify the returned metadata.

3.2.4. The responses section

The responses section does not add any items to the bundle metadata map. Instead, its offset and length are used in processing the index section (Section 3.2.1).

3.2.5. Starting from the end

The length of a bundle is encoded as a big-endian integer inside a CBOR byte string at the end of the bundle.

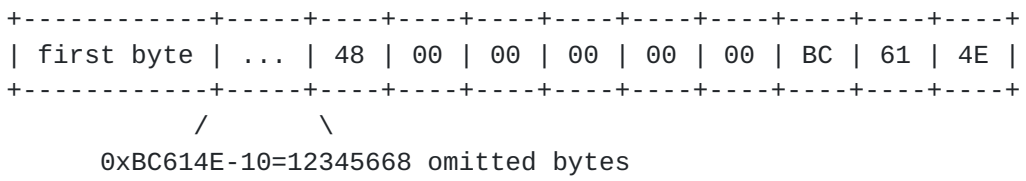

Figure 1: Example trailing bytes

Parsing from the end allows the bundle to be appended to another format such as a self-extracting executable.

To implement Section 2.2.1, taking a sequence of bytes "bytes", the client MUST:

1. Let "byteStringHeader" be "bytes[bytes.length - 9]". If "byteStringHeader is not "0x48` (the CBOR ([I-D.ietf-cbor-7049bis]) initial byte for an 8-byte byte string), return an error.

2. Let "bundleLength" be "[bytes[bytes.length - 8], bytes[bytes.length])" (the last 8 bytes) interpreted as a big-endian integer.

3. If "bundleLength > bytes.length", return an error.

4. Let "stream" be a new stream whose:

   * Available bytes are "[bytes[bytes.length - bundleLength], bytes[bytes.length])".

   * EOF flag is set.

   * Current offset is initially 0.

   * The seek to offset N and read N bytes operations succeed immediately if "currentOffset + N <= bundleLength" and fail otherwise.

5. Return the result of running Section 3.2 with "stream" as input.

3.3. Load a response from a bundle

The result of Load a bundle's metadata maps each request to a response, which consists of headers and a payload. The headers can be loaded from the bundle's stream before waiting for the payload, and similarly the payload can be streamed to downstream consumers.

response = [headers: bstr .cbor headers, payload: bstr]

To implement Section 2.3, the parser MUST run the following steps, taking the bundle's "stream" and one "request" and its "requestMetadata" as returned by Section 2.2.

1. Seek to offset "requestMetadata.offset" in "stream". If this fails, return an error.

2. Read 1 byte from "stream". If this is an error or isn't "0x82", return an error.

3. Let "headerLength" be the result of getting the length of a CBOR bytestring header from "stream" (Section 3.4.2). If "headerLength" is an error, return that error.

4. If "headerLength" is TBD or greater, return an error.

5. Let "headerCbor" be the result of reading "headerLength" bytes from "stream" and parsing a CBOR item from them matching the "headers" CDDL rule. If either the read or parse returns an error, return that error.

6. Let "headers"/"pseudos" be the result of converting "cbor-http-request" to a header list and pseudoheaders using the algorithm in Section 3.5. If this returns an error, return that error.

7. If "pseudos" does not have a key named ':status' or its size isn't 1, return an error.

8. If "pseudos[':status']" isn't exactly 3 ASCII decimal digits, return an error.

9. Let "payloadLength" be the result of getting the length of a CBOR bytestring header from "stream" (Section 3.4.2). If "payloadLength" is an error, return that error.

10. If "stream.currentOffset + payloadLength != requestMetadata.offset + requestMetadata.length", return an error.

11. Let "body" be a new body ([FETCH]) whose stream is a tee'd copy of "stream" starting at the current offset and ending after "payloadLength" bytes.

    TODO: Add the rest of the details of creating a "ReadableStream" object.

12. Let "response" be a new response ([FETCH]) whose:

* Url list is "request"'s url list,

* status is "pseudos[':status']",

* header list is "headers", and

* body is "body".

13. Return "response".

3.4. Parsing CBOR items

Parsing a bundle involves parsing many CBOR items. All of these items need to be canonically encoded.

3.4.1. Parse a known-length item

To parse a CBOR item ([I-D.ietf-cbor-7049bis]), optionally matching a CDDL rule ([I-D.ietf-cbor-cddl]), from a sequence of bytes, "bytes", the parser MUST do the following:

1. If "bytes" are not a well-formed CBOR item, return an error.

2. If "bytes" does not satisfy the core canonicalization requirements from Section 4.9 of [I-D.ietf-cbor-7049bis], return an error. This format does not use floating point values or tags, so this specification does not add any canonicalization rules for them.

3. If "bytes" includes extra bytes after the encoding of a CBOR item, return an error.

4. Let "item" be the result of decoding "bytes" as a CBOR item.

5. If a CDDL rule was specified, but "item" does not match it, return an error.

6. Return "item".

3.4.2. Parsing variable-length data from a bytestring

Bundles encode variable-length data in CBOR bytestrings, which are prefixed with their length. This algorithm returns the number of bytes in the variable-length item and sets the stream's current offset to the first byte of the contents.

To get the length of a CBOR bytestring header from a bundle's stream, the parser MUST do the following:

1. Let "firstByte" be the result of reading 1 byte from the stream. If "firstByte" is an error, return that error.

2. If "firstByte & 0xE0" is not "0x40", the item is not a bytestring. Return an error.

3. If "firstByte & 0x1F" is:

   0..23, inclusive Return "firstByte".

   24 Let "content" be the result of reading 1 byte from the stream. If "content" is an error or is less than 24, return an error.

   25 Let "content" be the result of reading 2 bytes from the stream. If "content" is an error or its first byte is 0, return an error.

   26 Let "content" be the result of reading 4 bytes from the stream. If "content" is an error or its first two bytes are 0, return an error.

   27 Let "content" be the result of reading 8 bytes from the stream. If "content" is an error or its first four bytes are 0, return an error.

   28..31, inclusive Return an error.

4. Return the big-endian integer encoded in "content".

3.5. Interpreting CBOR HTTP headers

Bundles represent HTTP requests and responses as a list of headers, matching the following CDDL ([I-D.ietf-cbor-cddl]):

```
headers = {* bstr => bstr}
```

Pseudo-headers starting with a ":" provide the non-header information needed to create a request or response as appropriate

To convert a CBOR item "item" into a [FETCH] header list and pseudoheaders, parsers MUST do the following:

1. If "item" doesn't match the "headers" rule in the above CDDL, return an error.

2. Let "headers" be a new header list ([FETCH]).

3. Let "pseudos" be an empty map ([INFRA]).

4. For each pair "name"/"value" in "item":

    1. If "name" contains any upper-case or non-ASCII characters, return an error. This matches the requirement in Section 8.1.2 of [RFC7540].

    2. If "name" starts with a ':':

        1. Assert: "pseudos[name]" does not exist, because CBOR maps cannot contain duplicate keys.

        2. Set "pseudos[name]" to "value".

        3. Continue.

    3. If "name" or "value" doesn't satisfy the requirements for a header in [FETCH], return an error.

    4. Assert: "headers" does not contain ([FETCH]) "name", because CBOR maps cannot contain duplicate keys and an earlier step rejected upper-case bytes.

        Note: This means that a response cannot set more than one cookie, because the "Set-Cookie" header ([RFC6265]) has to appear multiple times to set multiple cookies.

    5. Append "name"/"value" to "headers".

5. Return "headers"/"pseudos".

4. Guidelines for bundle authors

Bundles SHOULD consist of a single CBOR item satisfying the core canonicalization requirements (Section 3.4) and matching the "webbundle" CDDL rule in Section 3.1.

5. Security Considerations

Bundles currently have no mechanism for ensuring that the signed exchanges they contain constitute a consistent version of those resources. Even if a website never has a security vulnerability when resources are fetched at a single time, an attacker might be able to combine a set of resources pulled from different versions of the website to build a vulnerable site. While the vulnerable site could have occurred by chance on a client's machine due to normal HTTP caching, bundling allows an attacker to guarantee that it happens. Future work in this specification might allow a bundle to constrain its resources to come from a consistent version.

6. IANA considerations

6.1. Internet Media Type Registration

IANA maintains the registry of Internet Media Types [RFC6838] at https://www.iana.org/assignments/media-types [3].

- Type name: application
- Subtype name: webbundle
- Required parameters: N/A
- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See Section 5 of this document.
- Interoperability considerations: N/A
- Published specification: This document
- Applications that use this media type: None yet, but it is expected that web browsers will use this format.
- Fragment identifier considerations: N/A
- Additional information:
  - Deprecated alias names for this type: N/A
  - Magic number(s): 84 48 F0 9F 8C 90 F0 9F 93 A6
  - File extension(s): .wbn
  - Macintosh file type code(s): N/A
- Person & email address to contact for further information: See the Author's Address section of this specification.
- Intended usage: COMMON
- Restrictions on usage: N/A
- Author: See the Author's Address section of this specification.
- Change controller: The IESG iesg@ietf.org [4]

- Provisional registration? (standards tree only): Not yet.

6.2. Web Bundle Section Name Registry

IANA is directed to create a new registry with the following attributes:

Name: Web Bundle Section Names

Review Process: Specification Required

Initial Assignments:

Section Name	Specification	Metadata
"index"	Section 3.2.1	Yes
"manifest"	Section 3.2.2	Yes
"critical"	Section 3.2.3	Yes
"responses"	Section 3.2.4	No

Requirements on new assignments:

Section Names MUST be encoded in UTF-8.

Assignments must specify whether the section is parsed during Load a bundle's metadata (Metadata=Yes) or not (Metadata=No).

The section's specification can use the bytes making up the section, the bundle's stream (Section 2.1), the "sectionOffsets" CBOR item (Section 3.2), and the offset within the stream where sections start, as input, and MUST say if an error is returned, and otherwise what items, if any, are added to the map that Section 3.2 returns. A section's specification MAY say that, if it is present, another section is not processed.

7. References

7.1. Normative References

[appmanifest]
Caceres, M., Christiansen, K., Lamouri, M., Kostiainen, A., Dolin, R., and M. Giuca, "Web App Manifest", World Wide Web Consortium WD WD-appmanifest-20180523, May 2018, <https://www.w3.org/TR/2018/WD-appmanifest-20180523>.

[FETCH] WHATWG, "Fetch", June 2018, <https://fetch.spec.whatwg.org/>.

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-02 (work in progress), March 2018.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.

[INFRA] WHATWG, "Infra", June 2018, <https://infra.spec.whatwg.org/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SRI] Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", World Wide Web Consortium Recommendation REC-SRI-20160623, June 2016, <http://www.w3.org/TR/2016/REC-SRI-20160623>.

[URL] WHATWG, "URL", June 2018, <https://url.spec.whatwg.org/>.

7.2. Informative References

[I-D.yasskin-http-origin-signed-responses]
Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-03 (work in progress), March 2018.

[I-D.yasskin-webpackage-use-cases]
Yasskin, J., "Use Cases and Requirements for Web Packages", draft-yasskin-webpackage-use-cases-01 (work in progress), March 2018.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

7.3. URIs

[1] https://mailarchive.ietf.org/arch/search/?email_list=art

[2] https://github.com/WICG/webpackage

[3] https://www.iana.org/assignments/media-types

[4] mailto:iesg@ietf.org

Appendix A. Acknowledgements

Author's Address

Jeffrey Yasskin
Google

Email: jyasskin@chromium.org