Software is hard

More musings on software development
Jan Odvarko
odvarko@gmail.com

HAR 1.2 Spec

This document is intended to describe a **HTTP Archive 1.2** (frozen) format that can be used by HTTP monitoring tools to export collected data.

HTTP Archive v1.2

One of the goals of the HTTP Archive format is to be flexible enough so, it can be adopted across projects and various tools. This should allow effective processing and analyzing data coming from various sources. Notice that resulting HAR file can contain privacy & security sensitive data and user-agents should find some way to notify the user of this fact before they transfer the file to anyone else.

- The format described below is based on HTTP Archive 1.1.
- The format is based on JSON.
- Please follow-up in the newsgroup.
- An online HAR viewer is available.
- Report any problems in the issue list.
- See list of tools supporting HAR.

HAR Data Structure

HAR files are required to be saved in UTF-8 encoding, other encodings are forbidden. The spec requires that tools support and ignore a BOM and allow them to emit one if they like.

Summary of HAR object types:

- log
- creator
- browser
- pages
- pageTimings
- entries
- request
- response
- cookies
- headers
- queryString
- postData
- params

- content
- cache
- timings

<log>

This object represents the root of exported data.

```
{
    "log": {
        "version" : "1.2",
        "creator" : {},
        "browser" : {},
        "pages": [],
        "entries": [],
        "comment": ""
    }
}
```

- *version [string]* - Version number of the format. If empty, string "1.1" is assumed by default.
- *creator [object]* - Name and version info of the log creator application.
- *browser [object, optional]* - Name and version info of used browser.
- *pages [array, optional]* - List of all exported (tracked) pages. Leave out this field if the application does not support grouping by pages.
- *entries [array]* - List of all exported (tracked) requests.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

There is one <page> object for every exported web page and one <entry> object for every HTTP request. In case when an HTTP trace tool isn't able to group requests by a page, the <pages> object is empty and individual requests doesn't have a parent page.

<creator>

Creator and *browser* objects share the same structure.

```
"creator": {
    "name": "Firebug",
    "version": "1.6",
    "comment": ""
}
```

<browser>

```
"browser": {
    "name": "Firefox",
    "version": "3.6",
```

```
    "comment": ""
}
```

- *name [string]* - Name of the application/browser used to export the log.
- *version [string]* - Version of the application/browser used to export the log.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

<pages>

This object represents list of exported pages.

```
"pages": [
    {
        "startedDateTime": "2009-04-16T12:07:25.123+01:00",
        "id": "page_0",
        "title": "Test Page",
        "pageTimings": {...},
        "comment": ""
    }
]
```

- *startedDateTime [string]* - Date and time stamp for the beginning of the page load (ISO 8601 - YYYY-MM-DDThh:mm:ss.sTZD, e.g. 2009-07-24T19:20:30.45+01:00).
- *id [string]* - Unique identifier of a page within the <log>. Entries use it to refer the parent page.
- *title [string]* - Page title.
- *pageTimings[object]* - Detailed timing info about page load.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

<pageTimings>

This object describes timings for various events (states) fired during the page load. All times are specified in milliseconds. If a time info is not available appropriate field is set to -1.

```
"pageTimings": {
    "onContentLoad": 1720,
    "onLoad": 2500,
    "comment": ""
}
```

- *onContentLoad [number, optional]* - Content of the page loaded. Number of milliseconds since page load started (page.startedDateTime). Use -1 if the timing does not apply to the current request.
- *onLoad [number,optional]* - Page is loaded (onLoad event fired). Number of milliseconds since page load started (page.startedDateTime). Use -1 if the timing does not apply to the current request.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

Depeding on the browser, *onContentLoad* property represents **DOMContentLoad** event or **document.readyState == interactive**.

<entries>

This object represents an array with all exported HTTP requests. Sorting entries by *startedDateTime* (starting from the oldest) is preferred way how to export data since it can make importing faster. However the reader application should always make sure the array is sorted (if required for the import).

```
"entries": [
    {
        "pageref": "page_0",
        "startedDateTime": "2009-04-16T12:07:23.596Z",
        "time": 50,
        "request": {...},
        "response": {...},
        "cache": {...},
        "timings": {},
        "serverIPAddress": "10.0.0.1",
        "connection": "52492",
        "comment": ""
    }
]
```

- *pageref [string, unique, optional]* - Reference to the parent page. Leave out this field if the application does not support grouping by pages.
- *startedDateTime [string]* - Date and time stamp of the request start (ISO 8601 - YYYY-MM-DDThh:mm:ss.sTZD).
- *time [number]* - Total elapsed time of the request in milliseconds. This is the sum of all timings available in the timings object (i.e. not including -1 values) .
- *request [object]* - Detailed info about the request.
- *response [object]* - Detailed info about the response.
- *cache [object]* - Info about cache usage.
- *timings [object]* - Detailed timing info about request/response round trip.
- *serverIPAddress [string, optional]* (new in 1.2) - IP address of the server that was connected (result of DNS resolution).
- *connection [string, optional]* (new in 1.2) - Unique ID of the parent TCP/IP connection, can be the client or server port number. Note that a port number doesn't have to be unique identifier in cases where the port is shared for more connections. If the port isn't available for the application, any other unique connection ID can be used instead (e.g. connection index). Leave out this field if the application doesn't support this info.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

<request>

This object contains detailed info about performed request.

```
"request": {
    "method": "GET",
    "url": "http://www.example.com/path/?param=value",
    "httpVersion": "HTTP/1.1",
```

```
    "cookies": [],
    "headers": [],
    "queryString" : [],
    "postData" : {},
    "headersSize" : 150,
    "bodySize" : 0,
    "comment" : ""
}
```

- *method [string]* - Request method (GET, POST, ...).
- *url [string]* - Absolute URL of the request (fragments are not included).
- *httpVersion [string]* - Request HTTP Version.
- *cookies [array]* - List of cookie objects.
- *headers [array]* - List of header objects.
- *queryString [array]* - List of query parameter objects.
- *postData [object, optional]* - Posted data info.
- *headersSize [number]* - Total number of bytes from the start of the HTTP request message until (and including) the double CRLF before the body. Set to -1 if the info is not available.
- *bodySize [number]* - Size of the request body (POST data payload) in bytes. Set to -1 if the info is not available.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

The total request size sent can be computed as follows (if both values are available):

```
var totalSize = entry.request.headersSize + entry.request.bodySize;
```

<response>

This object contains detailed info about the response.

```
"response": {
    "status": 200,
    "statusText": "OK",
    "httpVersion": "HTTP/1.1",
    "cookies": [],
    "headers": [],
    "content": {},
    "redirectURL": "",
    "headersSize" : 160,
    "bodySize" : 850,
    "comment" : ""
 }
```

- *status [number]* - Response status.
- *statusText [string]* - Response status description.
- *httpVersion [string]* - Response HTTP Version.
- *cookies [array]* - List of cookie objects.
- *headers [array]* - List of header objects.
- *content [object]* - Details about the response body.

- *redirectURL [string]* - Redirection target URL from the Location response header.
- *headersSize [number]** - Total number of bytes from the start of the HTTP response message until (and including) the double CRLF before the body. Set to -1 if the info is not available.
- *bodySize [number]* - Size of the received response body in bytes. Set to zero in case of responses coming from the cache (304). Set to -1 if the info is not available.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

**headersSize* - The size of received response-headers is computed only from headers that are really received from the server. Additional headers appended by the browser are not included in this number, but they appear in the list of header objects.

The total response size received can be computed as follows (if both values are available):

```
var totalSize = entry.response.headersSize + entry.response.bodySize;
```

This object contains list of all cookies (used in <request> and <response> objects).

```
"cookies": [
    {
        "name": "TestCookie",
        "value": "Cookie Value",
        "path": "/",
        "domain": "www.janodvarko.cz",
        "expires": "2009-07-24T19:20:30.123+02:00",
        "httpOnly": false,
        "secure": false,
        "comment": ""
    }
]
```

- *name [string]* - The name of the cookie.
- *value [string]* - The cookie value.
- *path [string, optional]* - The path pertaining to the cookie.
- *domain [string, optional]* - The host of the cookie.
- *expires [string, optional]* - Cookie expiration time. (ISO 8601 - YYYY-MM-DDThh:mm:ss.sTZD, e.g. 2009-07-24T19:20:30.123+02:00).
- *httpOnly [boolean, optional]* - Set to true if the cookie is HTTP only, false otherwise.
- *secure [boolean, optional]* (new in 1.2) - True if the cookie was transmitted over ssl, false otherwise.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

<headers>

This object contains list of all headers (used in <request> and <response> objects).

```
"headers": [
    {
        "name": "Accept-Encoding",
        "value": "gzip,deflate",
        "comment": ""
```

```
    },
    {
        "name": "Accept-Language",
        "value": "en-us,en;q=0.5",
        "comment": ""
    }
]
```

<queryString>

This object contains list of all parameters & values parsed from a query string, if any (embedded in <request> object).

```
"queryString": [
    {
        "name": "param1",
        "value": "value1",
        "comment": ""
    },
    {
        "name": "param1",
        "value": "value1",
        "comment": ""
    }
]
```

HAR format expects NVP (name-value pairs) formatting of the query string.

<postData>

This object describes posted data, if any (embedded in <request> object).

```
"postData": {
    "mimeType": "multipart/form-data",
    "params": [],
    "text" : "plain posted data",
    "comment": ""
}
```

- *mimeType [string]* - Mime type of posted data.
- *params [array]* - List of posted parameters (in case of URL encoded parameters).
- *text [string]* - Plain text posted data
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

Note that *text* and *params* fields are mutually exclusive.

<params>

List of posted parameters, if any (embedded in <postData> object).

```
"params": [
    {
        "name": "paramName",
        "value": "paramValue",
        "fileName": "example.pdf",
        "contentType": "application/pdf",
        "comment": ""
    }
]
```

- *name [string]* - name of a posted parameter.
- *value [string, optional]* - value of a posted parameter or content of a posted file.
- *fileName [string, optional]* - name of a posted file.
- *contentType [string, optional]* - content type of a posted file.
- *comment [string, optional] (new in 1.2)* - A comment provided by the user or the application.

<content>

This object describes details about response content (embedded in <response> object).

```
"content": {
    "size": 33,
    "compression": 0,
    "mimeType": "text/html; charset=utf-8",
    "text": "\n",
    "comment": ""
}
```

- *size [number]* - Length of the returned content in bytes. Should be equal to response.bodySize if there is no compression and bigger when the content has been compressed.
- *compression [number, optional]* - Number of bytes saved. Leave out this field if the information is not available.
- *mimeType [string]* - MIME type of the response text (value of the Content-Type response header). The charset attribute of the MIME type is included (if available).
- *text [string, optional]* - Response body sent from the server or loaded from the browser cache. This field is populated with textual content only. The text field is either HTTP decoded text or a encoded (e.g. "base64") representation of the response body. Leave out this field if the information is not available.
- *encoding [string, optional]* (new in 1.2) - Encoding used for response text field e.g "base64". Leave out this field if the text field is HTTP decoded (decompressed & unchunked), than trans-coded from its original character set into UTF-8.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

Before setting the text field, the HTTP response is decoded (decompressed & unchunked), than trans-coded from its original character set into UTF-8. Additionally, it can be encoded using e.g. base64. Ideally, the application should be

able to unencode a base64 blob and get a byte-for-byte identical resource to what the browser operated on.

Encoding field is useful for including binary responses (e.g. images) into the HAR file.

Here is another example with encoded response. The original response is:

```
<html><head></head><body/></html>\n
"content": {
    "size": 33,
    "compression": 0,
    "mimeType": "text/html; charset=utf-8",
    "text": "PGh0bWw+PGhlYWQ+PC9oZWFkPjxib2R5Lz48L2h0bWw+XG4=",
    "encoding": "base64",
    "comment": ""
}
```

<cache>

This objects contains info about a request coming from browser cache.

```
"cache": {
    "beforeRequest": {},
    "afterRequest": {},
    "comment": ""
}
```

- *beforeRequest [object, optional]* - State of a cache entry before the request. Leave out this field if the information is not available.
- *afterRequest [object, optional]* - State of a cache entry after the request. Leave out this field if the information is not available.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

This is how the object should look like if no cache information are available (or you can just leave out the entire field).

```
"cache": {}
```

This is how the object should look like if the the info about the cache entry before request is not available and there is no cache entry after the request.

```
"cache": {
    "afterRequest": null
}
```

This is how the object should look like if there in no cache entry before nor after the request.

```
"cache": {
    "beforeRequest": null,
    "afterRequest": null
```

```
}
```

This is how the object should look like to indicate that the entry was not in the cache but was store after the content was downloaded by the request.

```
"cache": {
    "beforeRequest": null,
    "afterRequest": {
        "expires": "2009-04-16T15:50:36",
        "lastAccess": "2009-16-02T15:50:34",
        "eTag": "",
        "hitCount": 0,
        "comment": ""
    }
}
```

Both *beforeRequest* and *afterRequest* object share the following structure.

```
"beforeRequest": {
    "expires": "2009-04-16T15:50:36",
    "lastAccess": "2009-16-02T15:50:34",
    "eTag": "",
    "hitCount": 0,
    "comment": ""
}
```

- *expires [string, optional]* - Expiration time of the cache entry.
- *lastAccess [string]* - The last time the cache entry was opened.
- *eTag [string]* - Etag
- *hitCount [number]* - The number of times the cache entry has been opened.
- *comment [string, optional]* (new in 1.2) - A comment provided by the user or the application.

<timings>

This object describes various phases within request-response round trip. All times are specified in milliseconds.

```
"timings": {
    "blocked": 0,
    "dns": -1,
    "connect": 15,
    "send": 20,
    "wait": 38,
    "receive": 12,
    "ssl": -1,
    "comment": ""
}
```

- blocked [number, optional] - Time spent in a queue waiting for a network connection. Use -1 if the timing does not apply to the current request.

- dns [number, optional] - DNS resolution time. The time required to resolve a host name. Use -1 if the timing does not apply to the current request.
- connect [number, optional] - Time required to create TCP connection. Use -1 if the timing does not apply to the current request.
- send [number] - Time required to send HTTP request to the server.
- wait [number] - Waiting for a response from the server.
- receive [number] - Time required to read entire response from the server (or cache).
- ssl [number, optional] (new in 1.2) - Time required for SSL/TLS negotiation. If this field is defined then the time is also included in the connect field (to ensure backward compatibility with HAR 1.1). Use -1 if the timing does not apply to the current request.
- comment [string, optional] (new in 1.2) - A comment provided by the user or the application.

The *send*, *wait* and *receive* timings are not optional and must have non-negative values.

An exporting tool can omit the *blocked*, *dns*, *connect* and *ssl*, timings on every request if it is unable to provide them. Tools that can provide these timings can set their values to -1 if they don't apply. For example, *connect* would be -1 for requests which re-use an existing connection.

The *time* value for the request must be equal to the sum of the timings supplied in this section (excluding any -1 values).

Following must be true in case there are no -1 values (*entry* is an object in *log.entries*) :

```
entry.time == entry.timings.blocked + entry.timings.dns +
    entry.timings.connect + entry.timings.send + entry.timings.wait +
    entry.timings.receive;
```

Custom Fields

The specification allows adding new custom fields into the output format. Following rules must be applied:

- Custom fields and elements MUST start with an underscore (spec fields should never start with an underscore.
- Parsers MUST ignore all custom fields and elements if the file was not written by the same tool loading the file.
- Parsers MUST ignore all non-custom fields that they don't know how to parse because the minor version number is greater than the maximum minor version for which they were written.
- Parsers can reject files that contain non-custom fields that they know were not present in a specific version of the spec.

Versioning Scheme

The spec number has following syntax:

```
<major-version-number>.<minor-version-number>
```

Where the major version indicates overall backwards compatibility and the minor version indicates incremental changes. So, any backwardly compatible changes to the spec will result in an increase of the minor version. If an existing fields had to be broken then major version would increase (e.g. 2.0).

Examples:
`1.2 -> 1.3`

```
1.111 -> 1.112 (in case of 111 more changes)
1.5 -> 2.0 (2.0 is not compatible with 1.5)
```

So following construct can be used to detect incompatible version if a tool supports HAR since 1.1.

```
if (majorVersion != 1 || minorVersion < 1)
{
    throw "Incompatible version";
}
```

In this example a tool throws an exception if the version is e.g.: 0.8, 0.9, 1.0, but works with 1.1, 1.2, 1.112 etc. Version 2.x would be rejected.

HAR With Padding

Support for JSONP (JSON with padding) is not part of the core HAR spec. However, it represents very good feature for consuming HAR files online.

In order to server HAR files online the URL might include a callback URL parameter that should wrap HAR into a callback to a function (you might want to use *.harp extension for HAR files with padding).

```
http://www.example.com/givememyhar.php?callback=onInputData
```

Response for the URL above would be:

```
onInputData({
    "log": { ... }
});
```

See live example:

http://www.softwareishard.com/har/viewer/?inputUrl=http://www.janodvarko.cz/har/viewer/examples/inline-scripts-block.harp&callback=onInputData

```
<a href="http://www.softwareishard.com/har/viewer/?
inputUrl=http://www.janodvarko.cz/har/viewer/examples/inline-scripts-
block.harp&amp;callback=onInputData">http://www.softwareishard.com/har/viewer/?
inputUrl=http://www.janodvarko.cz/har/viewer/examples/inline-scripts-
block.harp&amp;callback=onInputData</a>
```

- *inputUrl* specifies an URL of the target HAR file (doesn't have to come from the same domain)
- *callback* species name of the function the HAR is wrapped in.

HAR Compression

Compression of the HAR file is not part of the core HAR spec. However, in order to store HAR files more efficiently, it is recommended that you compress HAR files on disk (you might want to use *.*zhar* extension for zipped HAR files).

Anyway, an application supporting HAR, is not required to support compressed HAR files. If the application doesn't support compressed HAR then it's the responsibility of the user to decompress before passing the HAR file into it.

HTTP Compression is one of the best practices how to speed up web applications and it's also recommended for HAR files.

Doc translation to Chinese provided by Jeremy.

Search:

Search

About Jan Odvarko

- About Me
- GitHub
- Google+
- LinkedIn
- Ohloh
- Quora
- Twitter

About My Projects

- Domplate Runner
- Firebug
- Firecookie
- HAR Adopters
- HAR Export Trigger
- HAR Viewer
- HTTP Archive Spec 1.2 (HAR)
- Web Socket Monitor

Domplate Tutorial

- Part I. Examples
- Part II. Examples

Extending Firebug Tutorial

- Part I. Hello World!
- Part II. Toolbar
- Part III. Options
- Part IV. Localization
- Part IX. Activable Panel
- Part V. Domplate
- Part VI. Yahoo! Search
- Part VII. Customize Net Panel
- Part VIII. Net Panel Listener
- Part X. Inspector
- Part XI. Infotip

- Part XII. Hello AMD!

Miscellaneous

- Extending Firebug
- Firebug Extensions
- Firebug Tips & Tricks

Subscribe

- Comments
- iPad
- Posts

Categories

- ConsoleExport
- Design Pattern
- Developer Tools
- Documentation
- Domplate
- Eventbug
- Extending Firebug Tutorial
- Extension Architecture
- Firebug
- Firebug Extension
- Firebug Lite
- Firebug Tip
- Firecookie
- Firestarter
- Fireunit
- HAR
- HTTP Monitor
- Memory Leaks
- NetExport
- Page Load Performance
- Pixel Perfect
- Planet Mozilla
- Prism
- React
- Release
- Selenium
- Testing
- Uncategorized
- WebSockets

Recent Tweets

The HAR Show: Capturing and Analyzing performance data with HTTP Archive format

http://t.co/q2N3U8zU
Firebug Tip: The Start Button
http://t.co/CY6IsFsg
#Firebug Tip: What the heck is BFCache?
http://t.co/OZNjSBkt
#Firebug Tip: Log Function Calls
http://t.co/WcdIVV8q
#Firebug Tip: Log DOM Events
http://t.co/dvrs5jdl