Web Archive Collection Zipped (WACZ)

Webrecorder Recommendation 03 June 2021

▼ **More details about this document**

This version:

https://specs.webrecorder.net/wacz/1.1.1/

Latest published version:

https://specs.webrecorder.net/wacz/latest/

Editors:

Ilya Kreymer (Webrecorder)

Ed Summers (Stanford University)

Additional Documents

Use Cases for Decentralized Web Archives

Previous versions

1.1.0

1.0.0

Repository

Github

Issues

Commits

Table of Contents

Abstract

WACZ is a media type that allows web archive collections to be packaged and shared on the web as a discrete file. A WACZ file includes all the data that is needed for the rendering archived content as well as contextual information required for users to interpret it. Rendering software can obtain this data on demand using HTTP Range requests, without requiring the entire file to be fully retrieved, or for it to be otherwise mediated by specialized server side software.

§ 1. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, and *SHOULD* 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.

§ 2. Status of This Document

This is a stable version of the WACZ standard and is in active use by the Webrecorder project. Please open GitHub issues for questions and suggestions.

§ 3. Terminology

This section defines the terms used in this specification and throughout web archives infrastructure. A link to these terms is included whenever they appear in this specification.

Collection
: An arbitrary set of related archived web pages and metadata based on some topic, website domain(s), time period, or other conceptual grouping.

Context
: Descriptive information about a web archive that helps a person using that web archive understand and interpret what the archive contains. This information can include why the content was selected for the archive, when it was created, who created it, and what tools or applications were used to create it.

IIPC
: The International Internet Preservation Consortium. An organization of libraries, archives and other organizations established in 2003 to coordinate efforts to preserve web content.

Media Type
: A two-part identifier for file formats that are transferred on the World Wide Web and the underlying Internet. [IANA-MEDIA-TYPES].

Package
: A file format that allows distinct files or bitstreams to be represented within it. Popular examples of packaging formats include ZIP, PDF, MP4, tar and Open Office XML.

Page
: A web document as viewed in a web browser that is viewing a specific URL. Sometimes referred to as a *web page*.

WACZ
: Web Archive Collection Zipped. A file that conforms to this specification which is used to package up WARC data and metadata into a ZIP file for distribution and replay on the web

WARC
: A file containing concatenated representations of web resources conforming to the [WARC] specification.

Wayback Machine
: A well known web application for replaying archived web pages that was initially developed at the Internet Archive and has been forked as an open soruce application by the IIPC.

Web Archive

A collection of files that preserve representations of web resources in the WARC format. A web archive may also include derivative files such as CDX indexes for accessing records within the archive.

ZIP file

A file conforming to the [ZIP] specification which is used to aggregate, compress, and encrypt files into a single interoperable container. WACZ allows for both ZIP and ZIP64 encodings for larger archives.

§ 4. Introduction

This specification defines a directory structure and ZIP format specification for sharing and distributing web archives. ZIP files using this format can be referred to as WACZ (Web Archive Collection Zipped).

§ 4.1 Motivation

The goal of this specification is to provide a portable format for web archives in order to achieve two broad goals for web archives:

1. *Social*: to provide an interoperable way of sharing web archive collections that includes the contextual information needed for users to interpret and meaningfully interact with them.

2. *Technical*: to provide an efficient way to dynamically load *small amounts of data* from a remotely hosted file on static storage, without requiring the entire file to be downloaded, or for the intervention of specialized server side applications.

To use and make sense of a web archive collection, it is necessary to have the archived web content as well as contextual information that describes what the collection contains as well as when and how it was created. The collection also requires a set of entry points or pages to use for browsing the collection.

All of this data needs to be packaged together so that the various pieces can be easily copied and transferred without accidentally separating them. This data package needs to to be easily transported from one storage system to another, sent as an attachment in an email, placed on a thumb drive, and hosted by simply serving it up at a given URL as a static document, possibly from cloud object storage, or a CDN.

Hosting web archives currently requires complex server infrastructure (e.g. a Wayback Machine) to serve WARC data in such a way that can be viewed in the browser. The WACZ format provides a storage approach optimized for efficient random-access to packaged up WARC data that allows the browser to render a page by fetching only what is needed for that particular page. This is done by leveraging the ZIP format's built-in index to locate the contents of the web archive and its constituent metadata.

WACZ is not designed to replace other web archiving formats. Rather it establishes a file packaging convention for all the data needed by a browser for efficient rendering of a web archive collection, and its contextualization.

§ 4.2 Existing Tools

The py-wacz repository contains a reference implementation for creating WACZ files from existing WARC files, and validating them. Parts of the specification are also implemented and in use by wabac.js and ReplayWeb.page.

§ 5. WACZ Object

A WACZ object consists of the following:

1. A `datapackage.json` file for recording technical and descriptive metadata specified in [FRICTIONLESS-DATA-PACKAGE].
2. An extensible directory and naming convention for web archive data.
3. A method for bundling the directory layout in a ZIP file.

§ 5.1 Directory Layout

A WACZ contains a directory structure, that contains web archive collection data which *MUST* conform to the [FRICTIONLESS-DATA-PACKAGE] specification. This directory structure looks like:

EXAMPLE 1

```
├── archive
│   └── data.warc.gz
├── datapackage.json
├── datapackage-digest.json
├── indexes
│   └── index.cdx.gz
└── pages
    └── pages.jsonl
```

§ 5.2 Directories and Files

§ 5.2.1 archive

The `archive` directory *MUST* contain one or more files in the [WARC] format. The files *SHOULD* use the `.warc` file extension unless they are GZIP encoded in which case they *MUST* use the `.warc.gz` file extension.

> EXAMPLE 2
>
> ```
> archive
> └── data.warc
> ```

§ 5.2.2 indexes

The `indexes` directory *MUST* include one or more indexes for the WARC data stored in `archive`. These index files allow clients to efficiently look up a URL to see if it is contained in the WACZ. Index files *MUST* contain CDXJ data and *MAY* be gzip compressed [PYWB-CDXJ].

> EXAMPLE 3
>
> ```
> indexes
> └── index.cdx.gz
> ```

§ 5.2.3 pages.jsonl

The `pages/pages.jsonl` *MUST* be present and include a list of 'Page' objects as [JSON-Lines] where each line *MUST* contain at least the following properties:

- `url` - a URL for the page
- `ts` - a [RFC3339] datetime string

Each entry in the [JSONL] file *MAY* contain the following properties to aid in navigating a web archive collection:

- `title` - a string describing the resource

- `id` - an arbitrary identifier for the resource
- `text` - text extracted from the snapshot
- `size` - an integer that represents the number of bytes for the page and all its resources

EXAMPLE 4

```
{"format": "json-pages-1.0", "id": "pages", "title": "All Pages"}
{"id": "1db0ef709a", "url": "https://www.example.com/page", "size": 1256, "ts": "2
{"id": "12304e6ba9", "url": "https://www.example.com/another", "size": 1256, "ts":
```

Each entry in the [JSONL] file *MAY* contain additional properties as long as they do not interfere with the required properties.

Other [JSONL] files *MAY* be added on using the same format in the `pages/` directory. A common use case is to include only the main pages in the `pages.jsonl`, while including additional pages, such as those discovered automatically via a crawl in an another file e.g. `extraPages.jsonl`.

§ 5.2.4 datapackage.json

The `datapackage.json` file *MUST* be present at the root of the WACZ which serves as the manifest for the web archive and is compliant with the [FRICTIONLESS-DATA-PACKAGE] specification. It *MUST* contain the following properties:

- `profile`: the string `data-package`
- `resources`: a list of file names, paths, sizes and fixity for all files contained in the WACZ.
- `wacz_version`: the version of WACZ used, for example `1.1.1`

EXAMPLE 5


```
{
  "profile": "data-package",
  "wacz_version": "1.1.1",
  "resources": [
     {
       "name": "pages.jsonl",
       "path": "pages/pages.jsonl",
       "hash": "sha256:8a7fc0d302700bed02294404a627ddbbf0e35487565b1c6181c729dff8d
       "bytes": 75
     },
     {
       "name": "data.warc",
       "path": "archive/data.warc",
       "hash": "sha256:0e7101316ba5d4b66f86a371ee615fbd20f9d3f32d32563ed2c829db062
       "bytes": 11469796
     }
  ]
}
```


The `datapackage.json` *SHOULD* include properties that allow rendering applications to present the user with contextual information about the web archive:

- `title`: a string or one sentence description for the collection
- `description`: a longer description of the archive's contents which *MUST* be Markdown formatted (plain text is valid Markdown)
- `created`: a [RFC3339] datetime for when the WACZ file was created
- `modified`: a [RFC3339] datetime for when the WACZ file was last modified
- `software`: A description of what software was used to create the WACZ file
- `mainPageUrl`: An optional URL of the main or starting page in the collection to be used for initial replay
- `mainPageDate`: An optional ISO-formatted date of the main or startng page in the collection to be used for initial replay

Other properties from the [FRICTIONLESS-DATA-PACKAGE] specification such as `licenses`, `version`, `organization`, `contributors`, `email` *MAY* be used. Custom properties that do not interfere with pre-existing properties *MAY* also be used.

5.2.5 datapackage-digest.json

A `datapackage-digest.json` file *SHOULD* be included in the root of the WACZ to verify the `datapackage.json` manifest with a hash and thus for the entire contents of the WACZ. If present the following properties *MUST* be included:

- `path`: the string "datapackage.json"
- `hash`: a cryptographic hash for the `datapackage.json` file

EXAMPLE 6

```
{
  "path": "datapackage.json",
  "hash": "sha256:ec1f44ab13e2c94b0ddf66e9673d585ba4a77e6f8c9cc30d8665da434557e885
}
```

For an approach to recording a cryptographic signature in the `datapackage-digest.json` in order to assert and prove the authorship of a WACZ please see WACZ Signing and Verification.

5.3 Other files and directories

Other files and directories *MAY* be present in a WACZ as long as they do not interfere with specified files and directories that are used by WACZ. Specifically, custom files and directories *MUST NOT* be added to the existing WACZ directories, `archive`, `indexes` and `pages`. Additional files *MUST* be listed in the resources section of `datapackage.json` to ensure conformance with [FRICTIONLESS-DATA-PACKAGE]

5.4 Zip Format

The entire directory structure *MUST* be stored in a standard [ZIP] file.

5.4.1 Zip Compression

Already compressed files *MUST NOT* be compressed again to allow for random access.

- All `archive/` files should be stored in ZIP with 'STORE' mode.
- All `index/*.cdx.gz` files should be stored in ZIP with 'STORE' mode.
- All files (`*.jsonl`, `*.json`, `*.idx`, `*.cdx`, `*.cdxj`) can be stored in the ZIP with either 'DEFLATE' or 'STORE' mode.

5.4.2 Zip Format File Extension

A ZIP file that follows this Web Archive Collection format spec *MUST* use the extension `.wacz`.

Such a file can be referred to as a WACZ file or a WACZ.

6. Processing Model

The [ZIP] file format provides efficient random access, which means archived web pages can be retrieved efficiently even from large web archive collections without requiring the entire WACZ to be transferred. To achieve this WACZ clients can read portions of the ZIP file on-demand using HTTP RANGE requests [RFC7233].

The processing model works as follows. Given a ZIP file, a client can quickly:

1. Read all entries to determine the contents of the ZIP file
2. Load collection metadata from the `datapackage.json`
3. Load a list of pages from `pages.jsonl`, if any

To lookup a given URL the client needs to:

1. Read the full CDX from ZIP
2. Binary search index looking for the URL
3. If a match found, get offset/length/location in WARC
4. Read compressed WARC chunk in ZIP

This approach is being used by ReplayWeb.page

7. Publishing

Because they are ZIP files WACZ can be hosted on the web as static files. This allows web archives to be easily maintained over time without relying on complex server side software, apart from widely available, open source, and well tested web server applications. If desirable WACZ files can be managed and made accessibile using HTTP

object stores available from cloud hosting providers, and content deliver networks that geographically position web-archives closer to their users. However there are certain considerations to make when publishing WACZ files.

§ 7.1 Content-Length

WACZ clients need to know how large an entire WACZ file is in order to download it prior to rendering, or to read it dynamically. To support this HTTP responses for WACZ files *MUST* use the `Content-Length` HTTP header.

§ 7.2 Partial Requests

Clients that render WACZ files typically need to be able to fetch content from the WACZ file on demand. For example when displaying archived content for a given URL that URL needs to be looked up in the CDXJ index, and the byte offsets from the index entry are then used to retrieve a portion of a given WARC file that is enclosed in the WACZ.

In order for clients to be able to perform this dynamic retrieval web servers that publish WACZ files *MUST* support HTTP range requests [RFC7233]. HTTP responses for WACZ HTTP requests *SHOULD* server WACZ files using the `Accept-Ranges` HTTP header.

§ 7.3 CORS

WACZ files and the their clients *MAY* be served from the same host name. However it can be useful to view the web archive from a host name that is distinct from the host name that is publishing the WACZ file. For example this is the case when publishing WACZ files using a cloud provider's HTTP object storage (e.g. `s3.amazonaws.com`) and making it viewable at another domain (`e.g. example.org`). It also is the case when WACZ publishers want to allow their web archives to circulate on the web, and be viewable in multiple locations.

For security reasons browsers restrict access to files hosted on a different domain than the websites that is trying to load them. In order to support loading from different domains WACZ files *SHOULD* be made available using the `access-control-allow-origin` [CORS] HTTP header.

§ 7.4 Media Type

WACZ HTTP responses for WACZ files *SHOULD* be published with the `application/wacz` media type.

7.5 Example Response

Given these requirements a minimal HTTP response for a WACZ could look like:

EXAMPLE 7

```
HTTP/2 200
Content-Type: application/wacz
Content-Length: 20961755
Accept-Ranges: bytes
Access-Control-Allow-Origin: *
```

A. References

A.1 Normative references

[CORS]
Cross-Origin Resource Sharing. Anne van Kesteren. W3C. 2 June 2020. W3C Recommendation. URL: https://www.w3.org/TR/cors/

[FRICTIONLESS-DATA-PACKAGE]
Frictionless Data Package. Paul Walsh; Rufus Pollock. Open Knowledge Foundation. 2 May 2017. URL: https://specs.frictionlessdata.io/data-package/

[IANA-MEDIA-TYPES]
Media Types. IANA. URL: https://www.iana.org/assignments/media-types/

[JSON-Lines]
JSON Lines: Documentation for the JSON Lines text file format. Ian Ward. 2 October 2013. URL: https://jsonlines.org/

[PYWB-CDXJ]
pywb Indexing: CDXJ Format. Webrecorder. URL: https://pywb.readthedocs.io/en/latest/manual/indexing.html#cdxj-index

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[RFC3339]
Date and Time on the Internet: Timestamps. G. Klyne; C. Newman. IETF. July 2002. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc3339

[RFC7233]

Hypertext Transfer Protocol (HTTP/1.1): Range Requests. R. Fielding, Ed.; Y. Lafon, Ed.; J. Reschke, Ed.. IETF. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7233.html

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

[WARC]

Information and documentation — WARC file format. International Organization for Standardization (ISO). August 2017. Published. URL: https://www.iso.org/standard/68004.html

[ZIP]

.ZIP File Format Specification. 15 July 2020. Final. URL: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT

↥