— Open Source WARC Tools —
Functional Requirements Specification

v1.0

Mark Middleton

Hanzo Archives Limited, Bonchurch Manor, Bonchurch Shute, Ventnor, Isle of Wight, PO38 1NU, United Kingdom

Registered address: 64 Clifton Street, London, EC2A 4HB, U.K.
Registered in England. Company No. 5410483. VAT No. 912 8708 19

T +44 78 0144 6473   F +44 87 0164 1643

markm@hanzoarchives.com

http://www.hanzoarchives.com

Table of Contents

Introduction

1. Background

The main goal of WARC Tools is to facilitate and promote the adoption of the WARC file format for storing web archives by the mainstream web development community by providing an open source software library, a set of command line tools, web server plug-ins and technical documentation for manipulation and management of WARC files.

WARC files are produced by web archiving crawlers, such as Heritrix — the open-source, extensible, web-scale, archiving quality web crawler developed by the Internet Archive with the Nordic National Libraries — and Hanzo's own commercial crawlers.

The project is lead by Hanzo Archives, in collaboration with the Internet Archive Web team, and supported by the International Internet Preservation Consortium (IIPC).

The core functionality of WARC Tools will be implemented as a library, and the functionality to be made available to end users external systems as command line tools, extensions to existing tools, and simple web applications for accessing WARC content. In addition the library will have APIs, Java and dynamic language bindings and will be made available as a software library for developers in these languages.

The library and tools will be scriptable (command lines in shell scripts, dynamic language bindings to the library), and programmable (dynamic language bindings, Java packages, and the C library itself).

Migration and interoperability with legacy tools is an important application for these tools and this project will implement functionality for legacy tools and the artefacts created by them — this will enable rapid progression and adoption of WARC by existing web archiving institutions and the mainstream web development community. To this end, integrating or linking to HTTrack, curl and wget for example are all going to be part of WARC Tools.

The library and tools will be implemented in ANSI C and will be highly portable, with build/installation on various Linux and Unix distributions, as well as Windows, together with unix man pages, build and installation guides, developer guides, etc.

The library and the tools will be open source, together with installers, documentation and man pages.

1.1. Resources

The project home page, including source code repository: http://code.google.com/p/warc-tools/

The project mailing list: http://groups.google.com/group/warc-tools

2. Scenarios and Use Cases

This section outlines a number of high-level real-world scenarios suggested by various IIPC member institutions and other interested parties — the requirements documented herein are largely derived from these, following this section.

2.1. Migration from Legacy Tools/Content to IIPC WARC

The table below describes scenarios for migration of pre-existing content to WARC files.

USE CASE	REQUIREMENTS
Migrate HTTrack Directory and HTTrack cache files to WARC	Collect HTTrack data from the directories output by HTTrack, and write to WARC files
Migrate wget mirror to WARC	Walk a wget mirror and write to WARC files.
Migrate curl mirror to WARC	Walk a curl directory and write to WARC files.
Add generic web content to WARC	Collect arbitrary web content, such as html files, images etc. (for example from a web server document root directory), and write to WARC files.
ARC to WARC	Command line script utilising ARC reader and WARC writer libraries. Can be easily adapted to a web UI if needed

2.2. Mass adoption of WARC

The table below describes scenarios for operational use of WARC files, implement the WARC standard.

USE CASE	REQUIREMENTS
Enable website mirroring tools to utilise WARC to store mirrored content	WARC writer library with specific modules for integrating or pipelining with HTTrack, wget and curl
Enable command line interrogation of WARC content using familiar command line tools	WARC reader library and command line tool to enable WARC content queries and extraction to be embedded in scripts, pipes, etc., and enable the same via a programmatic interface
Enable users to access website mirrors using browser access to WARC content	WARC reader library and mod_warc for Apache and Lighttp to provide browser access to WARC content via the familiar URL structure. Entry points and index generated by mod_warc on first access

USE CASE	REQUIREMENTS
Enable CD delivery of WARC content with installable WARC browser runtime	Packaged version of WARC reader library and mod_warc for Apache and Lighttp, together with installation scripts and binaries for major platforms, enables CD delivery of archived content
WARC Exploder	Independent tool and extension to wget and curl to extract content contained in an WARC file and store as static files in a directory. Optional re-writing for browsing.

2.3. Quality Assurance Aid

USE CASE	REQUIREMENTS
Enable easy programmatic access to WARC content	WARC reader library and mod_warc for Apache and Lighttp to provide browser access to WARC content via the familiar URL structure. Entry points and index generated by mod_warc on first access. API for automated browsing enabling test harnesses and automated testing.
Enable command line interrogation of WARC content using familiar command line tools	WARC reader library and command line tool to enable WARC content queries and extraction to be embedded in scripts, pipes, etc., and enable the same via a programmatic interface
Validation of WARC files	Enable identification and validation of WARC files using the file command and Jhove
Independent implementation of WARC standard	WARC Tools as a whole to be a new implementation, not utilising existing code in Heritrix and other Java-based tools, encouraging the wider Linux/C open source community to adopt WARC standards

3. Proposed Functional Deliverables

This project will develop and release the deliverables identified in the following table, that will conform to the functional requirements.

The non-functional deliverables resulting from this project are documented in the document “Non-Functional Requirements”.

Rows shaded green will be implemented in Phase I of the project; those shaded amber will be partially implemented to facilitate testing of the library in real world settings at the end of Phase I.

DELIVERABLE	DESCRIPTION
(W)ARC Writer	Library for writing crawl artefacts to ARC and WARC
(W)ARC Reader	Library for fast access to artefacts contained in ARC and WARC
WARC validator	Library for validating WARC files against a standard
(W)ARC Browser	Library expanding on (W)ARC Reader to facilitate web browsing of artefacts contained in (W)ARC files
Command line tools	Set of command line tools for manipulation of (W)ARC files and their content, exploiting the above libraries
mod_warc plug-in for Apache and lighttp web servers	Simple web container for (W)ARC files, providing web UI for searching contents and browser access to contents
Extensions for curl, wget and HTTrack	Add (W)ARC writing extensions and (W)ARC crawling/reading extensions to curl, wget and HTTrack
Unix file command/magic numbers	Contribute to the UNIX file command/magic number community, implementing the set of tests required to identify a WARC file
Jhove	Modules and Output Handlers for Jhove, for WARC identification, validation and characterisation
Dynamic language interface	Using SWIG, produce Python, Perl and Ruby interfaces for the library
Java interface	Produce a Java interface for the library
Various scripts	A number scripts to combine the functionality of the tools and the library for common applications.

Requirements

This section describes the functional requirements of the WARC Tools at a level of detail sufficient to outline their functional behaviour. In addition, it is hoped this document will confirm the ultimate goal of project — the development of a software framework for the management and manipulation of WARC files and their contents, together with a collection of tools and scripts to facilitate more mainstream adoption of WARC.

4. WARC writer/reader

The underlying framework of the WARC writer / reader is libwarc — a comprehensive, extensible, high-speed, general purpose framework for the management and manipulation of WARC files and their contents.

4.1. Universal interface

FR 1 — Any software tool or application based on libwarc shall require just a single header file.

Including this single header file in your software will enable access, modification and handling of all aspects of the WARC format:

```
    #include <warc.h>
```

FR 2 — The libwarc interfaces shall fully encapsulate and internal functionality, isolating by abstraction any tool or application based on libwarc from changes in the internal implementation.

The interfaces in libwarc will ensure that any further development of the framework, for example to follow developments in the WARC standard, will not impact any tool or application based on libwarc.

4.2. WARC writer interface

FR 3 — Libwarc shall provide a range of functions through a universal interface for creating each type of valid WARC-record, thus ensuring the resulting WARC files conform to the prevailing WARC standard

FR 4 — For each type of WARC-record, a set of functions shall be made available to create / modify the records properties.

For example, to create a "warcinfo" record:

```
    #include <warc.h>
    Warc w = Warc("foo.warc.gz");
    WarcInfo wi = NewWarcInfo();
    WarcInfoSetOperator(wi, "foo@bar.com");
    WarcInfoSetSoftware(wi, "heritrix/99.0.1");
    ...
    WarcRecordAppend(w, wi);
    ...
    DeleteWarcInfo(wi)[1];
    DeleteWarc(w);
```

4.3. WARC reader interface

FR 5 — Libwarc shall provide a range of functions through a universal interface for reading a range of valid WARC-records

Libwarc reader will thereby make read operations as simple as possible.

FR 6 — Libwarc shall provide a stable Application Programme Interface (API) to handle WARC-records as abstract objects (ADT)

Following the previous example, here is how to read a "warcinfo" record:

```
    #include <warc.h>
    Warc w = Warc("foo.warc.gz");
    WarcInfo wi = GetWarcInfo(w);
    char * op = WarcInfoGetOperator(wi);
    char * crawler = WarcInfoGetSoftware(wi);
    ...
    DeleteWarcInfo(wi);
    DeleteWarc(w);
```

4.4. ARC reader interface

FR 7 — Libwarc shall provide ARC-record reader functionality

Libwarc reader will thereby make read operations as simple as possible.

Following the previous example, here is how to read an ARC records "mime type" and "IP address" at position "offset":

```
    #include <warc.h>
    Arc a = Arc("bar.arc.gz");
    ArcRecord ar = ArcRecordAtOffset(a, offset);
    const char * ip = ArcRecordGetIP(ar);
    const char * mime = ArcRecordGetMimeType(ar);
    ...
    DeleteArcRecord(ar);
    DeleteArc(a);
```

[1] Note that these are simply functions for memory management and would be hidden in the libraries provided for Java, Python, etc.

4.5. Iterators

FR 8 — Libwarc shall provide a default iterator as a built-in function

Developers will be able to use the iterator to apply different types of filters to WARC files and handle them accordingly.

Libwarc will include three kinds of iterators:

- Records iterator
- MIMES iterator
- Custom iterator

FR 9 — Libwarc shall provide a records iterator to handle WARC-type records

An iterator on records (WARC-Type) will allow developers to extract and/or handle WARC-records of different block types. *Here is an example of use:*

```
    $ wfilter -i foo.warc.gz -t response
    $ wfilter -i bar.warc.gz -t continuation
    $ wfilter -i bar.warc.gz -t all (means all WARC-record)
```

FR 10 — Libwarc shall provide a MIMES iterator to handle MIME-types payloads

FR 11 — Iterators can use exact pattern search or regex expressions

An iterator on MIME types will provide exact pattern search or regex-based search on MIME types of payloads, as follows:

```
    $ wfilter -i foo.warc.gz -m "text/html"
    $ wfilter -i bar.warc.gz -m "image/(png|jpeg)"
    $ wfilter -i foo.warc.gz -m "image/*"
```

FR 12 — Libwarc shall provide an abstract interface for iterators, to enable custom iterators

Custom iterators will enable developers to define their own WARC iterator without changing any line of code. *Thus, software based on libwarc will be consistent in design and behaviour.*

FR 13 — Libwarc shall enable combinations of iterators to be used, i.e. composite iterators

Composite iterators enable developers to extend the filtering capacity of WARC by using the combination of iterators included by default. *Developers will be able to filter on various criteria:*

```
    $ wfilter -i foo.warc.gz -m "text/*" -t "conversion"
```

4.6. Controlled memory management

FR 14 — Libwarc shall be memory safe and manage dynamic memory internally

FR 15 — Libwarc shall use disk-based working memory

FR 16 — Dynamic memory management shall be hidden inside libwarc

The libwarc framework will provide internal routines for managing dynamic memory and therefore it will not require developers using libwarc to explicitly code for memory management.

Disk-based working memory will enable handling of WARC files of arbitrary size, only limited by disk size, not RAM.

4.7. Compression / Decompression of WARC-Records

Libwarc will perform all read and write operations on WARC files and records, including managing compression and decompression.

FR 17 — Libwarc shall use gzip as the default compression format

FR 18 — Libwarc shall provide a plug-in interface to enable use of alternative compression libraries, such as “gzip2”, “7zip”, etc.

FR 19 — Libwarc shall manage all write and read access

5. WARC Validator

Validity and consistency of the final library will be tested on a set of valid and non-valid WARCs in tests. Changes in the library and / or standard will be tested in regression tests.

FR 20 — WARC files shall be validated using a command line tool, “w-validator”, which will enable validation or rejection of WARC files.

Example:

```
    $ w-validator -i foo.warc.gz
```

6. WARC Browser

WARC Browser is a simple high level component that takes WARC files and associated CDX files and provides an API that allows browsing of those files via a suitable HTTP server.

FR 21 — WARC Browser shall provide an API that allows browsing of WARC files and associated CDX files via an HTTP server

FR 22 — WARC Browser shall support a rewriting interface

FR 23 — WARC Browser shall support a proxy-style interface

WARC Browser will be derived from existing Hanzo access technology, which has been in live service for nearly two years.

7. "mod_warc" Web server plugin

FR 24 — An Apache and Lighttp plug-in shall provide access to the libwarc API and WARC Browser over http

8. Command line tools exploiting libwarc

FR 25 — It shall be possible to convert ARC files to WARC files using a command-line tool called "arc2warc"

FR 26 — ARC to WARC conversions made by arc2warc shall be carried out according to a specification in a configuration file

arc2warc will be used as follow:

```
    $ arc2warc -i foo.arc.gz [-o bar.warc.gz] [-c config_file.cfg]
```

where : "config_file.cfg" is the configuration file containing information to direct the migration from ARC to WARC.

9. WARC extensions

Libwarc will enable WARC extensions for a variety of tools and scripts.

FR 27 — It shall be possible to collect HTTrack data from the directories output by HTTrack and write the data to WARC files

FR 28 — It shall be possible to walk a wget mirror and write the data to WARC files

FR 29 — It shall be possible to walk a curl directory and write the data to WARC files

FR 30 — It shall be possible to collect arbitrary web content, such as html files, images etc. (for example from a web server document root directory), and write the data to WARC files

FR 31 — A Python script shall be made available to enable rapid development of WARC based solutions

10. Command line tools extensions

FR 32 — WARC extensions shall be released as a patch to each of the commands "curl", "wget", and "httrack" projects

FR 33 — WARC extensions help content will be provided each of the commands "curl", "wget", and "httrack"

Example:

```
    $ curl --help | grep warc
    $ wget --help | grep warc
    $ py-warc --help
```

11. Identification information for UNIX "file" command

FR 34 — It shall be possible to identify WARC files using the unix “file” command

The project will contribute to the UNIX file command / magic number file community by implementing:

- The set of tests required to identify a WARC file
- Documentation for the file community
- Submission to UNIX and GNU Linux file distributions

See:

```
    $ man magic
    $ man file
```

12. Identification information for "Jhove" application

FR 35 — It shall be possible to identify and validate WARC files using “Jhove”

The project will deliver modules and output handlers for Jhove that perform:

- identification
- validation
    - well-formedness
    - valid
- characterisation

Together will appropriate documentation.

FR 36 — A Jhove plugin module and an output handler shall be made available for WARC files

FR 37 — WARC files in various test-states shall be provided that test the Jhove deliverables

13. Java and dynamic language interfaces

FR 38 — The C library shall be implemented to be compatible with the Simplified Wrapper and Interface Generator (SWIG, http://www.swig.org) — a wrapper for libwarc C code to allow it to be called natively in a wide variety of languages.

FR 39 — A Python interface shall be implemented

FR 40 — A Java interface shall be implemented

These implementations will allow the library to be used out of the box in Python and Java requiring no knowledge of the internal implementation of libwarc or C.

Despite the ease of providing a Java API to the C library this does present a possible risk to the standard itself. If most users of WARC depend on a single core implementation then there is a possibility that the specifics of the implementation override parts of the standard where variations are possible, and over time the dominance of a single implementation therefore may become a new **de-facto** standard overriding the official standard.

To mitigate this risk it is advantageous to develop a separate Java implementation as part of this project.

FR 41 — An independent Java implementation of libwarc may be implemented

FR 42 — The functionality of the library shall be exposed in such a way as to fit the metaphors and paradigms of the implementation language

For example in Java 1.5:

```
...
for (WarcRecord rec : new WarcFile
>> ("example-20070925092600.warc.gz")) {
>>   System.out.println(rec.getUrl());
>> }
```

and in Python:

```
for rec in WarcFile(  'example-20070925092600.warc.gz' ):
    print rec.getUrl()
```

Discussion

14. Change Log

This change log lists changes in reverse order, most recent changes towards the top.

14.1. Change Log v0.3 to v0.4

Removed non-functional requrements/deliverable in section 3 and referenced separate NFR document.

Slight amendment to FR 3.

14.2. Change Log v0.2 to v0.3

Reorder FR numbering from section 4.4 after adding “ARC Reader Interface” section

14.3. Change Log v0.1 to v0.2

This log contains significant changes to the document from v0.1 to v0.2.

14.3.1. Restore sequential numbering of requirements

Changed numbering of functional requirements to sequential, non-hierarchical numbering.

FR 9.1 -> FR 10

FR 10..33 -> FR 11..34

FR 33.1..33.2 -> FR 35..36

FR 34..38 -> FR 37..41

14.3.2. Better categorisation of Scenarios

Move “ARC to WARC” from section 2.2 to 2.1

Move “Independent implementation of WARC standard” from section 2.2 to 2.3