— Open Source WARC Tools —
Software 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

To aid understanding of these software requirements, it is recommended the read review the documents listed in the resources section above.

- 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

- Functional Requirements Specification: http://warc-tools.googlecode.com/files/warc_tools_frs.pdf
- Non-Functional Requirements: http://warc-tools.googlecode.com/files/warc_tools_nfr.pdf

1.2. Note on style

Original functional and non-functional requirements are referenced by their unique number, *e.g. FR 21*.

Helpful text is added to the specification to aid readability, but this text is not a requirement. Such text shall be in *italics*. For example, *this is not a requirement*.

Examples and code samples are in mono-spaced font, with a grey background. These are not requirements either.

Where the original requirement does not need elaboration, it will be shown here in full with its original unique number. This is a requirement.

All requirements are in brown, with a border, *for example*:

> EG 1 — This is a requirement

The software requirements specification in this document supersedes all previous requirements *and therefore will be the sole source of requirements for the remainder of the project*.

Functional Software Requirements

This section describes the software requirements of the WARC Tools at a level of detail sufficient for a software engineer to design and build the software, satisfying the functional requirements.

2. 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.

2.1. Universal interface

SRS 1 — There shall be a single entry point to libwarc, called "warc.h".

SRS 2 — The "libwarc" headers shall be structured in a hierarchical manner. The universal header "warc.h" will include all of them.

SRS 3 — It shall be possible for developers to access, modify and manipulate of all aspects of the WARC file format by including this single header file.

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

SRS 1-3 >> satisfies FR 1

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.

SRS 4 — The universal header "warc.h" shall ensure compatibility between all versions of libwarc.

This shall not need any additional user or developer requirements or action.

SRS 5 — The universal header "warc.h" shall ensure that deprecated function calls and changes to the library are notified at compile time.

SRS 6 — The interfaces in libwarc shall ensure that any changes to the library, will not affect any tool or application based on libwarc.

Typically, the library will be updated to follow developments in the WARC standard, for example.

SRS 4-6 >> satisfies FR 2

2.2. WARC writer interface

SRS 7 — The universal header "warc.h" shall provide normalised interfaces to enable developers to create valid and compliant WARC-records, based on the definition in the "ISO TC 46/SC 4 N 595"

standards document. The interfaces shall be made available to create WARC records of the following types:

- "warinfo"
- "response"
- "request"
- "metadata"
- "revisit"
- "conversion"
- "continuation"
- “resource”

SRS 7 >> satisfies FR 3

SRS 8 — Each WARC-record shall be accessible via a peer C class of the same name.

SRS 9 — The attributes of each WARC-record, as per the ISO standard specification, shall have a corresponding attribute in its peer C class.

SRS 10 — Each peer class shall expose a set a class functions to read, write, and update attributes for the corresponding WARC-record.

SRS 8-10 >> satisfies FR 4

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);
```

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

2.3. WARC reader interface

SRS 11 — Libwarc shall provide an API describing (1) the set of data, and (2) the set of operations that can be performed on the data. The data types shall be abstract (abstract data types — ADT), to ensure independence of concrete implementations.

SRS 12 — It shall be possible to create a WARC-record using a constructor, which will returns an abstract handle to data representing the WARC-record.

SRS 13 — It shall be possible to release the WARC-record using a destructor.

SRS 14 — Any operations on WARC-records shall be possible using functions accepting the abstract handle as an argument.

SRS 11-14 >> satisfies FR 5 and FR 6

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);
```

2.4. ARC reader interface

SRS 15 —Libwarc shall include ADT objects to handle read operations on ARC-records

SRS 15 >> satisfies FR 7

The ARC-record reader functionality is for data migration purposes. 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);
```

2.5. Iterators

SRS 16 — Libwarc shall provide a generic iterator, to enable the developer to iterate over all WARC-records and create an abstract WARC-document as a simple container

SRS 17 — Libwarc shall provide a WARC-record MIME-type iterator

SRS 18 — Libwarc shall provide a WARC-record-type iterator

SRS 19 — Libwarc's generic iterators may be customised for different purposes via callback handlers (i.e. hooks)

SRS 16-19 >> satisfies FR 8-12

Developers will be able to use the iterator to apply different types of filters to WARC files and handle WARC-records appropriately.

An iterator on records (WARC-Type) will allow developers to extract and/or handle WARC-records of different block types, for example:

```
$ 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)
```

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/*"
```

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.

SRS 20 — Libwarc's iterators may be combined into composite iterators to enable the developer to more than one search field

SRS 20 >> satisfies FR 13

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"
```

2.6. Controlled memory management

SRS 21 — Libwarc shall encapsulate and handle all memory management when processing WARC-records. In order for the memory management system to work properly, the developer must use the libwarc API correctly.

SRS 22 — Developers using libwarc shall not be required to allocate/release memory directly, instead the developer shall use libwarc's object constructor and destructor functions.

SRS 23 — Libwarc shall use dynamic heap memory for its internal usage.

SRS 24 — Libwarc shall allocate minimum memory heap to store WARC-record metadata.

SRS 25 — The payload (or the WARC-record document itself) are stored on disk, to avoid using memory heap, even for small objects.

SRS 26 — Libwarc shall not use file to memory mapping technology, instead libwarc will explicitly allocate memory as needed.

SRS 21-26 >> satisfies FR 14-16

Disk-based working memory will enable handling of WARC files of arbitrary size, only limited by disk size, not RAM. This technique is used in Hanzo's internal libraries which is able to handle 1 terabyte ARC-record.

2.7. Compression / Decompression of WARC-Records

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

SRS 27 — Libwarc shall support non-compressed WARC-records and compressed WARC-records and files

SRS 28 — The default compression format shall be Gzip

SRS 27-28 >> satisfies FR 17

SRS 29 — Libwarc shall support multiple compression schemas, loading a specific compressor at runtime as an external shared library.

This avoids dependencies on any specific compression format

SRS 30 — It shall not be possible to use more than one compression schema (including non-compression) within a single WARC file. (i.e. it is not possible to mix compression schemes within a single WARC file).

SRS 29-30 >> satisfies FR 18

3. WARC Validator

SRS 31 — A command line tool shall be implemented utilising libwarc to check the consistency of WARC-records and their conformance to the WARC ISO standard.

SRS 32 — The command line tool shall notify the user of any WARC-record's anomalies, missing required fields or incompatible fields types.

SRS 32 >> satisfies FR 20

Example:

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

4. WARC Browser

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

SRS 33 — Libwarc shall provide a set of classes to enable remote management of WARC-records

SRS 34 — It shall be possible to perform read operations (read from offset, filters, etc.) on WARC-records from a remote location via http.

SRS 35 — *For security reasons*, it shall not be possible to perform write or update operations on a WARC-record remotely

SRS 36 — WARC browser shall not support CDX files because the CDX file format is not a standard at this time and is outside of scope.

SRS 33-36 >> satisfies FR 21

Different systems may implement different CDX file formats, for example IA Wayback machine CDX format is different from Hanzo Enterprise CDX format. Moreover, Hanzo handle two CDX file formats at the same time for performance reasons.

Additionally, the CDX file format belongs to the old ARC file format and needs to be redesigned for the new WARC standard.

We recommend FR 21 is removed from scope in the FRS.

SRS 37 — WARC Browser shall support a client-side rewriting interface by using javascript code to rewrite links being delivered alongside archived content. This is based on the principles implemented in the Wayback Machine.

SRS 37 >> satisfies FR 22

SRS 38 —A web proxy interface shall be implemented that allows the user to set their web browser proxy to the one provided by the interface and thereby ensure all content is delivered from the archive and not from the live web.

SRS 38 >> satisfies FR 23

5. "mod_warc" Web server plugin

SRS 39 — Libwarc shall be incorporated within an Apache module to enable all actions specified in SRS 34-36 to be executed within Apache.

SRS 40 — Libwarc shall be incorporated within a Lighttp module to enable all actions specified in SRS 34-36 to be executed within lighttp.

SRS 39-40 >> satisfies FR 24, NFR 14

6. Command line tools exploiting libwarc

SRS 41 — A command line tool “arc2warc” incorporating libwarc shall be able to migrate data in ARC-records to WARC-record format.

SRS 42 — The default operation of "arc2warc" shall carry out a one-to-one mapping of record fields, by converting each ARC-record to a corresponding "response" WARC-record and "metadata" WARC-record, which shall include information about the conversion process.

SRS 43 — "arc2warc" shall have make a default operation in cases where an ARC-record has no corresponding field in the WARC-record.

SRS 41-43 >> satisfies FR 25

SRS 44 — It shall be possible to specify non-default operations of "arc2warc" using a named configuration file, which will describe the desired ARC-record to WARC-record conversion.

SRS 44 >> satisfies FR 26

For example, the configuration file may specify the ”continuation” option, to limit the size of any WARC-record to, say, 100MB, in which case ”arc2warc” will split any ARC-record with size exceeding 100MB into a number of WARC-records, each with size <= 100MB.

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 specified in SRS 44.

7. WARC extensions

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

SRS 45 — A set of command line tools incorporating libwarc shall perform migration of "HTTrack" archives to WARC-records.

SRS 46 — The HTTrack archive file format and link strategy may vary from version to version of HTTrack, therefore it shall be possible to adapt the migration scripts to deal with these changes.

SRS 45-46 >> satisfies FR 27

SRS 47 — A set of command line tools incorporating libwarc shall perform migration of "wget" archives to WARC-records.

SRS 47 >> satisfies FR 28

SRS 48 — A set of command line tools incorporating libwarc shall perform migration of "curl" archives to WARC-records.

SRS 48 >> satisfies FR 29

SRS 49 — A set of command line tools and an API incorporating libwarc shall enable the collection of online documents, such as html and embedded files, etc., and write them to valid WARC- records.

This feature is not intended as a substitute for client side archiving.

SRS 50 — The command line tools and API in SRS 50 will not include any links extraction features.

SRS 49-50 >> satisfies FR 30

SRS 51 — Python scripts shall be implemented incorporating libwarc, and making all of the functionality of libwarc and API available in Python.

SRS 51 >> satisfies FR 31

8. Command line tools extensions

SRS 52 — Extensions to "HTTrack", "wget" and "curl" incorporating libwarc shall be provided as patches to recent and specific versions of each tool, to enable users of the tool to access functionality of libwarc

SRS 53 — Helper documentation for libwarc functionality shall be made available within the "HTTrack", "wget" and "curl" commands.

SRS 52-53 >> satisfies FR 32-33

Example:

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

```
$ py-warc --help
```

9. Identification information for UNIX "file" command

SRS 54 — A magic number for WARC shall be created and incorporated in the “file” mime-type database, enabling the simple identification of WARC files via the Unix "file" command

SRS 54 >> satisfies FR 34

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*

10. Identification information for "Jhove" application

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

SRS 55 — The WARC validator tool specified in SRS 31-32 shall be extended to optionally make use of the Jhove command line API to identify and validate WARC files, *i.e. given a specific WARC file, this command shall be able to identify the file as a WARC file, validate the level of compliance with a given standard in terms of well-formedness and validity, and finally to characterise the file by extracting and displaying significant properties contained in the file.*

SRS 55 >> satisfies FR 35

SRS 56 — WarcMdoule and WarcHandler plugin modules shall be implemented for Jhove Plugin layer to enable identification and validation of WARC files.

SRS 57 — WARC files in various test-states shall be provided that test the Jhove deliverables

SRS 56-57 >> satisfies FR 36-37

11. Java and dynamic language interfaces

SRS 58 — Void, duplicate of SRS 79

SRS 59 — Libwarc shall provide interfaces to SWIG wrappers to allow dynamic language bindings (Python, Ruby, Perl, Lua ...)

SRS 58-59 >> satisfies FR 38, NFR 12, NFR 18

SRS 60 — A Python interface to libwarc shall be implemented using the SWIG wrapper

SRS 60 >> satisfies FR 39, NFR 19

SRS 61 — A Java interface to libwarc shall be implemented using the SWIG wrapper and/or JNI

SRS 61 >> satisfies FR 40, NFR 20

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.

SRS 62 — An independent Java implementation of libwarc may be implemented subject to review of deliverables satisfying SRS 61

SRS 62 >> satisfies FR 41

SRS 63 — Libwarc and the bindings to its functionality shall enable the use of libwarc's iterators described in SRS 16-20 to be used within various dynamic languages and in Java v1.4 and earlier, using metaphors and paradigms familiar to those languages.

SRS 64 — Libwarc and the bindings to its functionality shall enable the use of libwarc's iterators described in SRS 16-20 to be used within Java v1.5 and and later, using Java's new container iterators, such as "for" and "foreach".

SRS 63-64 >> satisfies FR 42, NFR 20

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()
```

Non-Functional Software Requirements

This section describes the software requirements of the WARC Tools at a level of detail sufficient for a software engineer to design and build the software, satisfying the non-functional requirements and which may guide the architectural shape of the solution and the project as a whole.

12. Technical Requirements

SRS 65 — It shall be possible for libwarc to be able to handle WARC file of any size, with minimal memory usage.

SRS 66 — It shall be possible for libwarc to be able to read, write and update WARC files at high speed, *i.e. the design should focus on minimum memory footprint and performance.*

SRS 65-66 >> satisfies NFR 1

SRS 67 — Libwarc shall be implemented to WARC v0.17.

SRS 68 — Major revisions to the WARC ISO standard may lead to revisions of libwarc, but these revisions should be isolated and not affect any tools developed incorporating libwarc

SRS 67-68 >> satisfies NFR 2

12.1. Installation scripts and tools

SRS 69 — Libwarc shall be developed on GNU/Linux, Fedora 7

SRS 70 — Libwarc shall be shipped with a manual and build scripts

SRS 71 — Libwarc shall be shipped with installation guides for Fedora, Debian, FreeBSD, Mac OS X 10.5 and Windows XP.

SRS 72 — Libwarc shall be shipped with a developer guide and useful examples.

SRS 73 — Libwarc shall be shipped with a number of ready-to-use command lines, *for example, warcdump (to dump the contents of a WARC file), w-filter, etc.*

SRS 69-73 >> satisfies NFR 3, 6, 7

12.2. Unix command-line style tools

SRS 74 — Utility and application level functionality of WARC Tools shall be made available to end users as command line tools, extensions to existing tools, and simple web applications for accessing WARC content

SRS 74 >> satisfies NFR 4

12.3. Community release and documentation

SRS 75 — Communication and support shall be provided to the open source community. This support must be provided for at least term of the project.

SRS 75 >> satisfies NFR 5

SRS 76 — Patches implemented for third party projects shall be contributed and distributed to the appropriate community

SRS 76 >> satisfies NFR 8

NFR 9 — The code and documentation shall be licensed using an open source license.

SRS 77 — The code and documentation of libwarc shall be licensed to the community using an IIPC approved license, such as Apache 2.0 or BSD licence

SRS 77 >> satisfies NFR 9

SRS 78 — Libwarc and associated tools shall be shipped with installers to "mod_warc" for Apache (v2.X) and Lighttpd (v1.4.X) servers

SRS 78 >> satisfies NFR 10

12.4. Orthogonal technology

SRS 79 — Libwarc shall be implemented in C and shall conform to ANSI-C standard C99 revision.

SRS 80 — To ensure code portability on older computer architectures, Libwarc shall be compatible with any compiler conforming to the ANSI-C standard C89 revision.

SRS 79-80 >> satisfies NFR 11

SRS 81 — Command line tools incorporating libwarc shall be atomic — i.e. each tool performing a single function, performing it perfectly, and in the spirit of Unix command lines tools. These may be combined using pipes and redirection and scripting, to create more high level commands.

SRS 81 >> satisfies NFR 13

SRS 82 — Libwarc source code shall be released in the following archives: "libwarc-version.tar.gz", "libwarc-version.tar.bz2", "libwarc-version.zip" together with their corresponding digests. Developers may then adapt the build configuration files for best performance on their target machines.

SRS 83 — Libwarc binary modules, ready to use binary commands and libraries (both static and shared), compiled with generic optimisation flags, shall be released also

SRS 82-83 >> satisfies NFR 15

SRS 84 — Libwarc shall be made available as a binary for at least GNU/Linux, FreeBSD, Mac OS X and Windows XP.

SRS 85 — Libwarc shall compile and run under Solaris, AIX, MingW or other Unix compliant system and may compile and run under Windows XP.

SRS 86 — Libwarc shall not depend on a specific build tool in order to be built from source. However, makefiles for each target shall be included to simplify deployment.

SRS 87 — The default compiler used for all Unix systems to build libwarc will be gcc v.3.4.4 (or above). For Windows, the default compiler will be Microsoft Visual C++ 6.X (or above).

SRS 84-87 >> satisfies NFR 16

SRS 88 — Only essential external libraries shall be used in libwarc, such as Gzip compression and wide characters encoding libraries.

SRS 89 — Assembly code and specific system features shall not be used in libwarc to ensure resulting code is widely portable across multiple target architectures

SRS 88-89 >> satisfies NFR 17

Discussion

13. Notes on phase I/II implementation split

Hanzo will endeavour to deliver certain SRS functionality in Phase I. However this may be subject to change, depending on architecture and design decisions, testing needs and priorities. It should be noted that many of these features will not be fully implemented in Phase I.

However, any partial implementations will be completed during Phase II.

The preliminary list is as follows:

- SRS 1-26, 31-32
- SRS 41-44 (limited prototypical implementation for testing)
- SRS 65-71
- SRS 74
- SRS 79-84
- SRS 86-89

14. Changes v0.3 to v1.0

Added WARC files explicitly in SRS 27

Removed SRS 58 (duplicate of SRS 79)

Added clarifying note on SRS 66

Added clarifying note on SRS 84-85 regarding Windows support

15. Changes v0.2 to v0.3

Amended section 13, notes on phase I/II implementation split:

- ‘community priorities’ changed to ‘priorities’
- SRS 1-32 changed to SRS 1-26, 31-32 (compression handling will be phase II)
- Qualified phase I implementation of SRS 41-44 as without it, this may be impossible

16. Changes v0.1 to v0.2

SRS 7 — added “resource” record type

Added “Notes on phase I/II implementation split” to Discussion section

Completed missing SRS’s