warc-tools version 0.17 A library for data archiving

warc-tools version 0.17 A library for data archiving

Table of Contents

List of Examples

Chapter 1. introduction

The warc-tools is a library containing a set of functions that allow to store data and retrieve them from WARC files. A WARC file is an archive file stored in the WARC (Web ARChive) format. It is a concatenation of several data block where each one is encapsulated with a text header containing information about the data block itself and/or about the WARC file containing it. An encapsulated data block is then called a WARC Record. Thus, with the warc-tools library, we have access to functions that will permit us to create WARC Records from a given data file that we want to archive, and also to store it in an existing or a new WARC file. In addition, we have access to an other kind of functions that will allow us to retrieve WARC Records from an existing WARC file and to extract the data stored in it.

Also, this library allows us to access and read file stored respecting the old ARC archiving format, from which the WARC format was born. It offers functions to access records stored in such files and retrieve the data blocks they contain too.

In the following, we give a description of every application and function offered with the warc-tools to the user in order to archive data in the WARC format, to retrieve WARC Records stored in a WARC file, or, if he has archives in the ARC format, we describe how to use our own ARC files reader to get their data.

This document is composed into three parts :

- How to use warc-tools : describes how to use warc-tools applications, this is the only part you need you just know how to run these programs.
- Detailed utilisation : describes in more details the utilisation of the applications by simple examples.
- Programming with the warc-tools library : describes how to use libwarc functions which the library provides for development.

Chapter 2. How to use warc-tools

This chapter contains a copy of warc-tools man page, and nothing else.

2.1. NAME

- warcdump - dumps a WARC file
- arc2warc - ARC to WARC converter
- arc2warc.sh - converts all WARC file in directory to WARC file
- warcfilter - filters WARC Record based on Uri, content type or record type
- warcvalidator - checks WARC file consistency
- warcvalidator.sh - checks if all WARC file in directory are valid
- warcserver - starts the WARC server
- warcclient - to access remote WARC resource

2.2. SYNOPSIS

- `warcdump -f {file.warc} [-v] [-t {working_dir}]`
- `arc2warc -a {file.arc} -f {file.warc} [-c] [-t {working_dir}]`
- `arc2warc.sh -d {dirname} [-b] [-c] [-t {working_dir}] [-v] [-h]`
- `warcfilter -f {file.warc} [-u {uri}] [-m {mime}] [-r {rtype}]`
  `[-v] [-t {working_dir}]`
- `warcvalidator -f {file.warc} [-v] [-t {working_dir}]`
- `warcvalidator.sh -d {dirname} [-t {working_dir}] [-v] [-h]`
- `warcserver -i {ip} -p {port} -x {prefix} [-s {name}] [-t {working_dir}]`
- `warcclient -i {ip} -p {port} -t {remote_warc} -o {local_warc} -s {server_name}`
  `[-f {offset}] -[u {uri_pattern}] [-n {content_type_pattern}]`
  `[-d {record_type_string}] [-c] [-r]`

2.3. DESCRIPTION

warcdump : lists the header's fields of all WARC Records in a WARC file.

warcdump looks down the records of WARC file one by one, extracts header fields of the current record and displays their values in the screen.

arc2warc : converts an ARC file into a WARC file.

arc2warc creates a new WARC file initially empty, looks down the records of the ARC file one by one, extracts current ARC record, converts it into a WARC Record and stores this last one in a WARC file.

arc2warc.sh : converts all ARC files into WARC files in a directory by calling the arc2warc command described previously for each ARC file in the current directory.

warcfilter : lists like warcdump the headers fields of WARC Records present in a WARC file but only those that are corresponding to the filter value. The filter can be used on the WARC-Subject-Uri field (if existing), on the Content-Type field or on the WARC-Type field.

warcvalidator : checks if a WARC file is valid or not. A WARC file must follow a specific grammar. This grammar can be found the following address :
http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml

warcvalidator.sh : checks if all WARC files in a directory are valid or not. It calls the warcvalidator command described previously for each WARC file in the directory.

warcserver : a WARC server that allows to satisfy requests on WARC files contained on a specific directory. there are four kinds of requests:

- A whole WARC file transfer request.
- A lonely WARC Record transfer request.
- A filtered WARC file transfer request.
- A WARC file Records headers listing request.

warcclient : allows to send request to a server for claiming warc resources. You can make request to the WARC server described above or to Apache or lighttpd server but you must check that mod_apache or mod_lighttpd are correctly configured in server machine. Read the `"/doc/install"` for more details

2.4. OPTIONS

1. **warcdump** :

```
-f    : valid WARC file name
[-v]  : dump ANVL (default false)
[-t]  : temporary working directory (default ".")
```

2. **arc2warc**

```
-a    : valid ARC file name
-f    : valid WARC file name
[-c]  : WARC file will be GZIP compressed (default no)
[-t]  : temporary working directory (default ".")
```

3. **arc2warc.sh**

```
-d     : directory name containing ARC files
-c     : WARC files will be GZIP compressed
         (default no)
-t     : temporary working directory (default ".")
-h     : print this help message
-v     : output version information and exit
```

4. **warcfilter**

```
-f    : valid WARC file name
[-u]  : compare with URI
[-m]  : compare with MIME
[-r]  : compare with record types
  (see "public/wrectype.h" for possible values)
[-v]  : dump ANVL (default no)
[-t]  : temporary working directory (default ".")
```

5. **warcvalidator**

```
-f    : valid WARC file name
[-v]  : verbose mode (default no)
[-t]  : temporary working directory (default ".")
```

6. **warcvalidator.sh**

```
-d     : directory name containing WARC files
-t     : working temporary directory (default ".")
-h     : print this help message
-v     : output version information and exit
```

7. **warcserver**

```
[-i]  : ip address
[-p]  : port number
[-x]  : directory prefix
[-s]  : server name (default to "iipc")
```

```
[-t]  : temporary working directory (default ".")
```

8. **warcclient**

```
[-i]  : ip address
[-p]  : port number
[-s]  : server name
[-u]  : filter file on Record uri value
        (default no)
[-n]  : filter file on content type value
        (default no)
[-d]  : filter file on Record type value
        (default unknown)
[-t]  : remote WARC filename
[-o]  : output WARC filename
[-f]  : WARC offset (default 0)
[-r]  : get all the WARC file (default no)
```

Chapter 3. Detailed utilisation

This chapter gives more details on the use of warc-tools applications.

3.1. warcdump

The option -f is mandatory when we use warcdump. It is used to indicate the name of the WARC file to dump.

The option -t is optional it allows to give the application working directory for temporary files creation. By default working directory is the current one.

The option -v is optional. It specifies if the user wants to also get anvl fields details with the header.

Her, there is two usage examples, in the first one only the header is dumped, in the second both of the header and the anvl fields are dumped:

Example 3-1. How to use warcdump command to get only the header.

```
users@users-desktop:~$ warcdump -f file.warc -t /tmp/
```

Example 3-2. How to use warcdump command to get the header and the anvl fields.

```
users@users-desktop:~$ warcdump -f file.warc -t /tmp/ -v
```

3.2. arc2warc

The option -a is mandatory when we use arc2warc. It is used to indicate the name of the ARC file to convert.

The option -f is mandatory. It must be followed by the name of the output WARC file.

The option -c is optional. If you use -c, the resulting WARC file is generated compressed following the gzip format. By default, the WARC file is not generated commpressed. The WARC file mode of compression is independent of ARC file mode of compression, the user has to choose the method of compression.

The option -t is optional it allows to give the application work directory for temporary files creation. By default worker directory is the current one.

For the generation of an uncompressed WARC file `"file.warc"` from an ARC file `"file.arc"`, we can use the following command shown in this example :

Example 3-3. How to use the arc2warc command for the conversion of an ARC file into an uncompressed WARC file.

```
users@users-desktop:~$ arc2warc -a file.arc -f file.warc -t /tmp/
```

For the generation of a compressed WARC file `"file.warc.gz"`"from the same ARC file :

Example 3-4. How to use the arc2warc command for the conversion of an ARC file into a compressed WARC file.

```
users@users-desktop:~$ arc2warc -a file.arc -f file.warc -t /tmp/ -c
```

arc2warc.sh is shell script used to convert all ARC files in a directory to WARC files. The option of this command are similar to the arc2warc options, except that the option -d is added to indicates the directory where the ARC file are stored. We do not use -f and -a options in this case. The resulting WARC files will have the same names of their origin ARC file but with the extension ".warc" instead of ".arc". In the case when we reclaim a compressed output, the extension ".gz" will be added to the end of each WARC file name.

An example of utilisation of this command while passing the directory `"/tmp/file"` as input is :

Example 3-5. How to use arc2warc.sh command.

```
users@users-desktop:~$ arc2warc.sh -d /tmp/file -t /tmp/
```

3.3. warcfilter

The option -f is mandatory when we use warcfilter. It is used to indicate the name of the WARC file to filter. The option -v and -t are as described previously.

The option -u is to be used if we want to apply its argument as a filter on the WARC-Target-URI fields, if existing, of the WARC Records.

The option -m is to be used if we want to apply its argument as a filter on the Content-Type fields of the WARC Records.

The option -r is to be used if we want to apply its argument as a filter on the WARC-Type fields of the WARC Records. which represents the filter.

In The following example, we dump headers fields of a WARC file using warcfilter, passing the filter "http:" to be applied to uri field.

Example 3-6. How to use warcfilter command (usage of filter on WARC-Target-URI fields).

```
users@users-desktop:~$  warcfilter -f file.warc -u http: -t /tmp/
```

3.4. warcvalidator

The option -f is mandatory when you use warcvalidator. It is used to indicate the name of the WARC file to validate.

The option -t is optional, it is as described previously.

The option -v is different from -v options given previously, in this case it allows to display a result message.

The following example shows how to use the warcvalidator commad to test the validity of the WARC file `"file.warc"`:

Example 3-7. How to use the warcvalidator command.

```
users@users-desktop:~$ warcvalidator -f file.warc  -v
```

warcvalidator.sh is a script shell that can validate several WARC files in a directory. It calls the warcvalidator command defined previously. The option -d in this case is used to give the directory of the WARC files. The option -v in this case only specifies the warc version.

To validate all WARC files in the directory `"/tmp/file"`, we can proceed as in the following example :

Example 3-8. How to use the warcvalidator.sh command.

```
users@users-desktop:~$  warcvalidator.sh -d /file/tmp
```

3.5. warcserver

The options -i ,-p, -x are mandatory when we use warcserver command.

The option -i specifies the listening IP address. When its value is 0.0.0.0 you can listen from any IP address.

The option -p allows to define the listening port. In Unix systems we must have the permissions to listen port bellow 1024.

The option -x allows to define directory where WARC files are stored. The server will look for the WARC files required by the clients in this directory.

The option -t is optional, it allows to specify working directory.

The option -s is optional, it is used to define a name for the server. If you do not use this option the server name by default is "iipc".

Here, an example which shows how to start server on port 8080 that listens from any IP address. The WARC files in this server are in the directory `"/home/warcfile"` and we name the server by *`"warc_server"`*

Example 3-9. How to use the warcserver command.

```
users@users-desktop:~$  warcserver -i 0.0.0.0 -p 8080 -x /home/warcfile \
                                -t /tmp/ -s  warc_server
```

3.6. warcclient

The option -i -p and -s are mandatory, the client makes request to the located server at the address specified as argument of the -i option, on the port specified after -p. the option -s indicates for which kind server the request is sent to: the server may be a warcserver (launched by the command warcserver), an apache2 server with a mod_warc module, a lighttpd-cgi server using a warc.cgi module or a lighttpd-fcgi server using a warc.fcgi module.

- If you connect to a warcserver (launched by the command warcserver on the same IP address and port), the server name in this case is warcserver
- If you want to connect to Apache server (configuration of Apache server is shown in section mod_apache) server name in this case is apache.

- If you want connect to lighttpd server (Configuration of lighttpd server is show in section mod_lighttpd) server name in this case is lighttpd-cgi or lighttpd-fcgi (for cgi and fastcgi support respectively).

The option -t and -o are mandatory, the option -t must preceed the name of WARC file that the client want to get from server. Option -o must preceed the file name desired in the client machine.

The option -f gives the value of the offset in warc file from where we start the transfer. The offset is the beginning of the WARC Record to get in WARC file. If you do not pass -f to the command offset by default is zero.

The option -r is used to indicate that the whole WARC file is required, and the transfer will start from the indicated offset.

The result obtained when you want to get file or record are not filtered. If you want to have filtered WARC files, you can use options -u, -n or -d.

i. Use the option -u to filter on the WARC_TARGET_URI field, it comes before the string which represents the filter.

ii. Use the option -n to filter on the Content-Type field, it comes before the string which represents the filter.

iii. Use the option -d to filter on WARC-Type field, it comes before the string which represents the filter. If you specifies nothing after -d the value of filter is "unknown".

iv. Use the option -l to get the list of the WARC Redords stored in the required WARC file. In this case, the output file will be used to store this list which can be written into four possible formats: html, xml, text and json. each element of the list will give some information on the concerned WARC Record like its rank, offset, its header, etc.

There are examples that show how to use warcclient command

Example 3-10. How to use the warcclient to get a WARC Record.

```
users@users-desktop:~$ warcclient  -i  192.168.1.6 -p 8080 -s warcserver \
                                -t  test.warc -o vertest.warc  -f 0
```

In this case, we are sending a request to a warcserver, at IP address 192.168.1.6 whith the listening port 8080, to get record that begins at the offset 0 in the WARC file test.warc. The name of the returned file is vertest.warc in the client machine.

Example 3-11. How to use the warcclient to get a WARC file.

```
users@users-desktop:~$ warcclient  -i  192.168.1.6 -p 8080 -s apache2  \
                                -t test.warc -o vertest.warc  -f 0 -r
```

This request is similar to previous one, but here we send it to an apache2 server with a mod_warc module, and we want to get the whole WARC file from offset 0.

Example 3-12. How to use the warcclient to get a filtered WARC file.

```
users@users-desktop:~$ warcclient  -i  192.168.1.6 -p 8080 -s warcserver\
                                -t  test.warc -o vertest.warc  -f 0 -n gif
```

This request is similar to previous requests but in this case we want to get the WARC file from offset 0 including only the WARC Records whose Content-Type field contains the string **"gif"**.

Example 3-13. How to use warcclient to get the records list.

To get the lising of the WARC Records stored in the wanted WARC file using an apache2 server, we may proceed like the following. Her, we've chose the **xml** output format. It is possible also to use **html**, **text** and **json**

```
users@users-desktop:~$ warcclient  -i  192.168.1.6 -p 8080 -s apache \
                                -t  test.warc -o vertest.warc  -f 0 -l xml
```

In the same way, we can use mod_lighttpd by using supported cgi or fastcgi modules by typing in the case where we want to use cgi:

Example 3-14. How to use warcclient with lighttpd server using cgi module.

```
users@users-desktop:~$ warcclient -i 192.168.1.6 -p 8080 -s lighttpd-cgi \
                                -t  test.warc -o vertest.warc  -f 0 -l xml
```

And in the case where we want to use fastcgi:

Example 3-15. How to use warcclient with lighttpd using fastcgi module.

```
users@users-desktop:~$ warcclient -i 192.168.1.6 -p 8080 -s lighttpd-fcgi \
                                 -t  test.warc -o vertest.warc  -f 0 -l xml
```

3.7. mod_apache

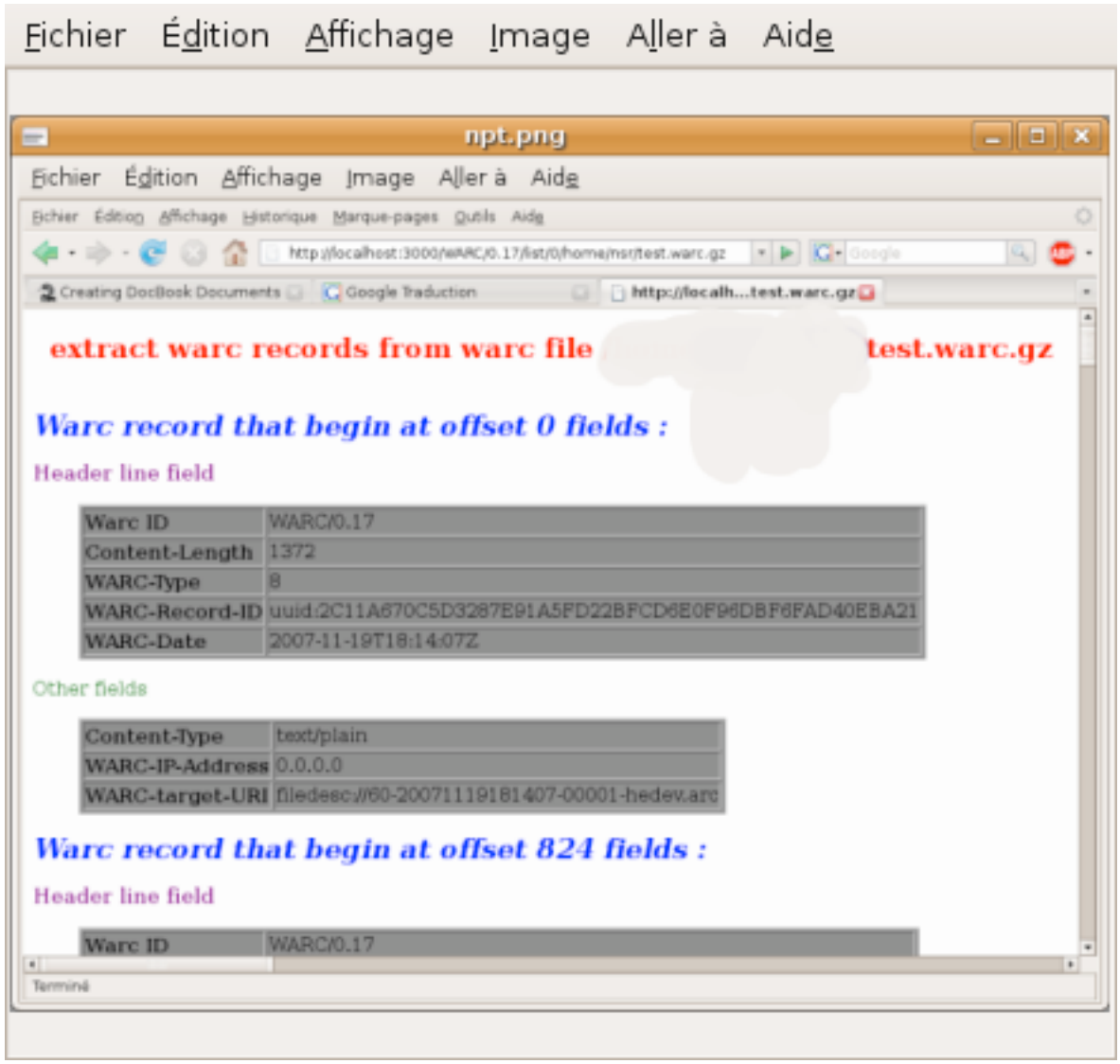
To know how to peroform the installtion and the configuration of an apache server to make run the mod_warc module, we can read the section reserved to mod_apache in file `"doc/install"` in the warc-tools source code.

We can type Url in browser to get AWRC resources. Url must follow a rest format. This format is explained in file `"doc/install"`

If we start an apache server in localhost at port 3000, we can type this url to list headers and anvl fields of the WARC Records from WARC file `"home/nsr/test.warc.gz"` from offset 0.

Example 3-16. How to send a request to an apache 2 server with the browser using the the

mod_warc module.



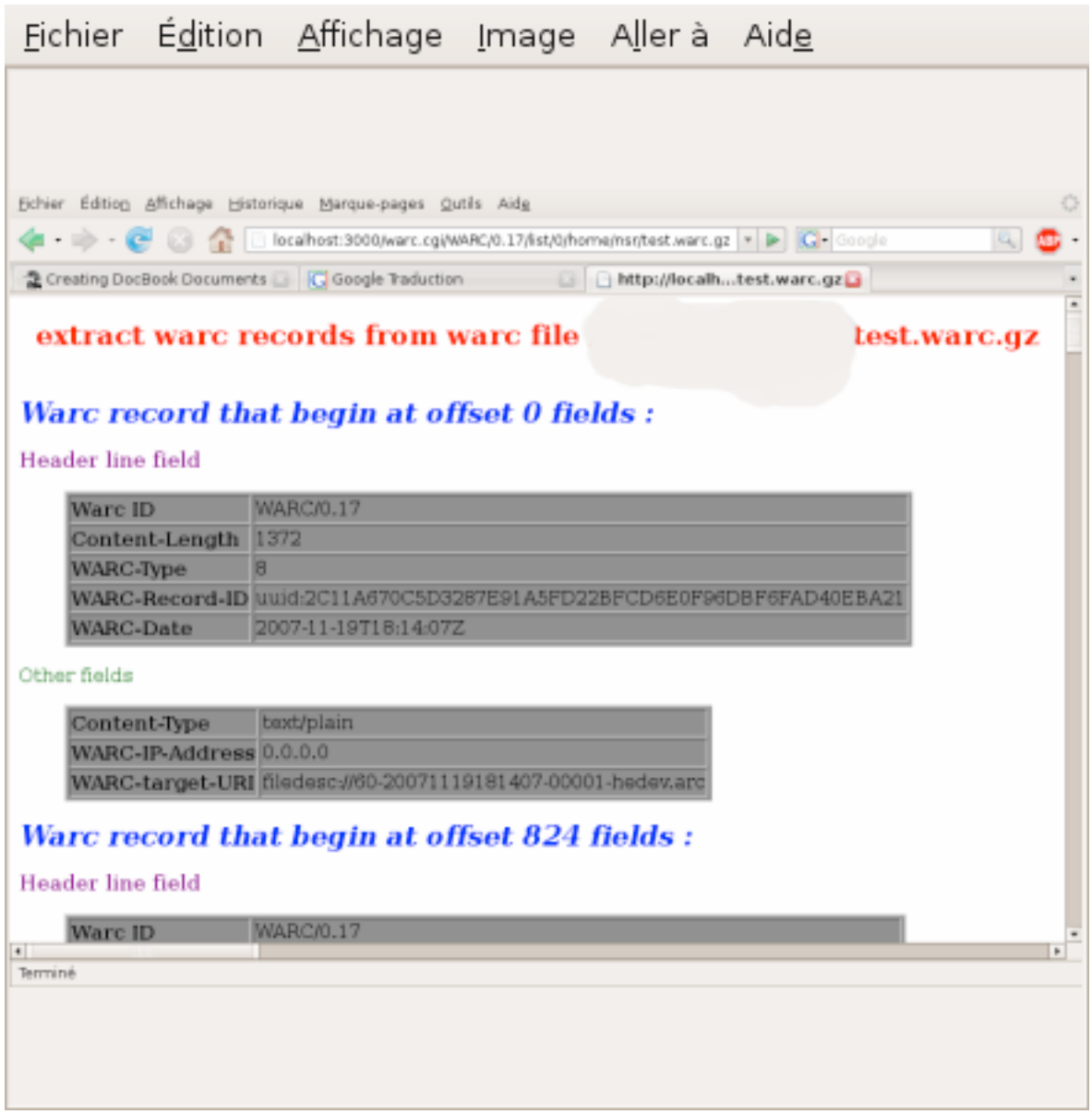
3.8. mod_lighttpd

To know how to peroform the installation and the configuration of a lighttpd server to make either warc.cgi or warc.fcgi moules run, we can read the section reserved to mod_lighttpd in the file `"doc/install"` in warc source code.

If we use a lighttpd server, we can use mod cgi and mod fastcgi to get warc resources. In this case, we use an url in the rest format but we have to precede this url by the wanted script cgi or fastcgi.

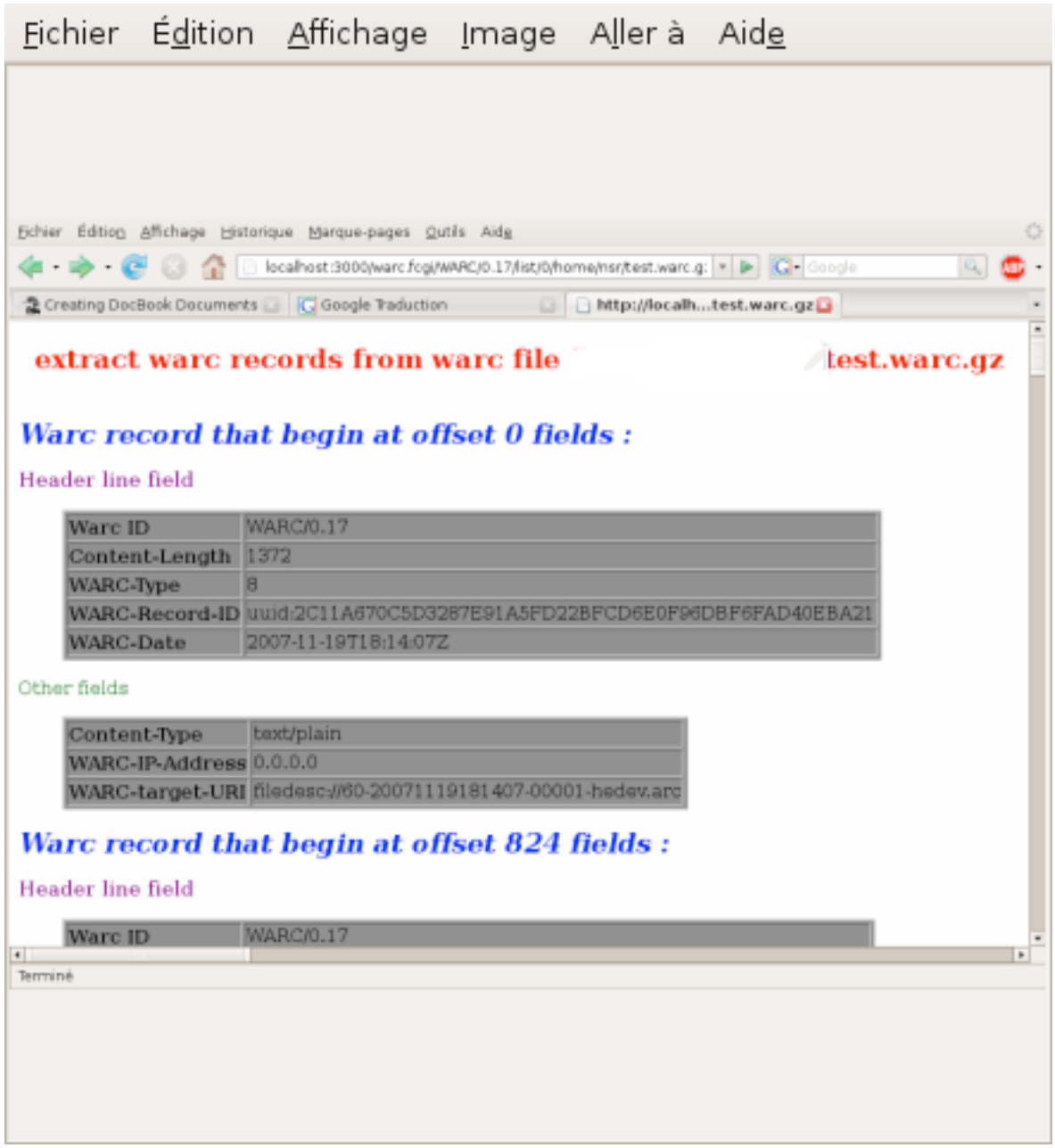
If we start a lighttpd server in localhost at port 3000, we can type this url to list the headers and anvl fields of the WARC Records from a WARC file `"home/nsr/test.warc.gz"` by using cgi script.

Example 3-17. How to send a request to a lighttpd server with a browser using warc.cgi script.



In the same way, we can type the same url to list the headers and anvl fields of the WAR Records from a WARC file `"home/nsr/test.warc.gz"` by using fastcgi script.

Example 3-18. How to send a request to a lighttpd server with a browser using the fastcgi.



Chapter 4. Programming with the warc library

This chapter describes the principals of the functions privided for programming with the libwarc. However for more details we can refer to the documentation generated with Doxygen from the warc-tools source code.

The warc library has been designed for three main purposes :

- The warc library provides a set of routines that allow WARC files creation, this routines must be able to check that the arguments given by the user are conform with the WARC format.
- The warc library provides a set of routines that allow to extract information from WARC files (headers, anvl fields and data block). These routines must be able to check that the data obtained is conform with the WARC format (validation of a WARC file).
- The warc library provides a set of routines that allow to convert files from the Arc format to the WARC format. These routines must be able to check that the ARC file is conform with the ARC format.

We can use the routines offered by the warc library for other purposess. In the following, we will describe the main data types and functions of this library.

4.1. Generalities

4.1.1. Object-oriented concept

The way how the source code is written makes the warc function very easy to use. The project is built by using the oriented-object concept in C language. Although the C language is not an object-oriented language, we can simulate the object-oriented beahaviour by using mainly pointers to functions concept. For using all functions of the warc library, we must include the header file `"warc.h"` in the program.

We use the function `bless` to create any object. In order to differentiate between the classes, the first argument of this function is the name of object. For example, if we want to create a WFile object, the name of this parameter is *`WFile`*.

The function `bless` returns the pointer to othe created object in the case of the success of creation.

The function `destroy` is used to destroy any object created by `bless`. It takes as parameter the pointer of the object.

This way to work notably facilitates the memory management, every object created by `bless` must be destroyed by `destroy`.

4.1.2. The user interface

The warc library provides to the functions user a clear and easy interface : A WFile class for the manipulation of WARC files, a WRecord class for the manipulation of the WARC Records, the ARecord object for the manipulation of ARC Records and finally the class Afile for exploitation of the ARC files. The prototypes of all theese function can be found in the corresponding header files in the `/lib/public/` directory.

4.2. Data types

The warc library has defined its own data types to make the information manipluation esay. Here there are those that the user will have to handle.

warc_bool_t

This is the warc boolean type having two possible values :*WARC_TRUE* and *WARC_FALSE*.

warc_i32_t

This is a simple signed integer, we prefer use this type to get protability with any architecture.

warc_u32_t

This is a simple unsigned integer, we prefer use this type to get protability with any architecture.

warc_i8_t

This is a simple signed character, we prefer use this type to get protability with any architecture.

warc_u8_t

This is a simple unsigned character, we prefer use this type to get protability with any architecture.

warc_i64_t

This is a simple signed long intreger, we prefer use this type to get protability with any architecture.

warc_u64_t

This is a simple unsigned long integer, we prefer use this type to get protability with any architecture.

warc_compt_t

This is an enumerated type used to specify compression mode for the WARC files, three values are possible : *WARC_FILE_UNCOMPRESSED*, *WARC_FILE_COMPRESSED_GZIP*, *WARC_FILE_COMPRESSED_GZIP_DEFAULT_COMPRESSION*, *WARC_FILE_COMPRESSED_GZIP_NO_COMPRESSION*, *WARC_FILE_COMPRESSED_GZIP_BEST_SPEED*, *WARC_FILE_COMPRESSED_GZIP_BEST_COMPRESSION*, and *WARC_FILE_DETECT_COMPRESSION*

warc_mod_t

This is an enumerated type used to specify the mode of the openening of a WARC file object. There are two possible modes *WARC_FILE_READER*, *WARC_FILE_WRITER*.

warc_rec_t

This an enume used to specfies the value of the WARC-TYPE field. Nine values are possible *WARC_UNKNOWN_RECORD* , *WARC_INFO_RECORD*, *WARC_RESPONSE_RECORD* , *WARC_REQUEST_RECORD*, *WARC_METADATA_RECORD* , *WARC_REVISIT_RECORD*, *WARC_CONVERSION_RECORD*, *WARC_CONTINUATION_RECORD* and *WARC_RESOURCE_RECORD*

4.3. Creation and destruction of objects

4.3.1. Creation and destrunction of WFile object

We use function `bless` to create a WFile object for the manipulation of a WARC file

```
void * bless (WFile, const char* fname, warc_u32_t  maxsize,
           wfile_mod_t mode, const wfile_comp_t compressed ,
           const char * dname, const warc_u32_t dname_len)
```

`maxsize` is a warc_u32_t paraleter guiving the maximum size of the WARC file. Used only in the writing case.

`mode` is wfile_mod_t parameter indicating the opening mode of the WARC file.

`compressed` is a wfile_comp_t parameter indicating the compression mode of the WARC file (useful in writing mode).

In general *`WARC_FILE_DETECT_COMPRESSION`* is used to allow autodetection of the compression mode in the reading mode.

`fname` and `dname` are simple strings guiving respectively the name of the WARC file and the working directoy of the WFile objet routines (for temporary files creation). `dname_len` is the length of the `dname` string.

If we want to get information from an existing WARC file `"file.warc"` present in some directory, we create the WFile object in reading mode like this :

Example 4-1. Creation of a WFile object in reading mode

```
bless (WFile, "file.warc", 0, WARC_FILE_READER,
    WARC_FILE_DETECT_COMPRESSION, ".")
```

`maxsize` in not used in this case because its value is not important. We notice also that when a WFile is created in reading mode, the file descriptor inside it will automatically seek the begining of the WARC file.

If you want to open a WaRC file object to append int it some WARC Records, we must create a WFile object in writng mode like this :

Example 4-2. Creation of a WFile object in writing mode

```
bless (WFile, "file.warc", 600* 1024 * 1024,
    WARC_FILE_WRITER,WARC_FILE_DETECT_COMPRESSION, ".")
```

We use *WARC_FILE_WRITER* as opening mode in this case. In this mode, the value of the maxsize field is significant: it indicates the maximum size of the WARC file that must not be exceeded when we write in the WARC file. Notice that if this value is below the currennt size of the file, the WFile object is not created. This allows to avoid the resize of the WARC file and hence the loose of the data stored in. When a WFile object is created in writing mode, its internal file descripto will automatically seek the end of the WARC file, this will ensure that a WARC Record will never be stored in the middle, and hence corrupt the WARC file, but it is allways appended in the end

To destroy cleanly an already created WFile object, we may use the function `destroy`. It will take as parameter the pointer to the WFile object instance we want to free.

```
destroy (void * refobject).
```

4.3.2. Creation and destruction of a WRecord object

We may use `bless` and `destroy` to create and destroy WARC Record object. The function `bless` takes no parameter for the creation of a *WRecord* object (a WARC Record object).

```
void * recobject = bless (WRecord);
```

To destroy cleanly an already created WRecord object, we may use the function `destroy`. It will take as parameter the pointer to the ARecord object instance we want to free.

```
destroy (void * refobject).
```

4.3.3. Creation and destrunction of AFile object

We use function `bless` to create a AFile object for the manipulation of a ARC file.

```
void * bless (AFile, const char* fname, const afile_comp_t compressed , const char * dn
```

`compressed` is a afile_comp_t parameter indicating the compression mode of the ARC file.

In general *`ARC_FILE_DETECT_COMPRESSION`* is used to allow autodetection of the compression mode.

`fname` and `dname` are simple strings guiving respectively the name of the WARC file and the working directoy of the WFile objet routines (for temporary files creation). `dname_len` is the length of the `dname` string.

To destroy cleanly an already created AFile object, we may use the function `destroy`. It will take as parameter the pointer to the AFile object instance we want to free.

```
destroy (void * refobject).
```

4.3.4. Creation and destruction of an ARecord object

We may use `bless` and `destroy` to create and destroy ARC Record object. The function `bless` takes no parameter for the creation of a *`ARecord`* object (an ARC Record object).

```
void * recobject = bless (ARecord);
```

To destroy cleanly an already created ARecord object, we may use the function `destroy`. It will take as parameter the pointer to the ARecord object instance we want to free.

```
destroy (void * refobject).
```

4.4. WFile object routines

In this section, we give a description of the main functions provided to manipulate a WFile object after having created it with `bless`. A WARC File have to be only manipulated through a WFile object using its routins. That will allow the safe recovering of the information stored inside. It is dangerous and not recomanded to manipulate a WARC file with an other way at the risk of corrupting it.

4.4.1. Reading the WARC Records of a WARC file

To read in sequence the WARC Records of a WARC file opned by the construction of a WFile object on it, we have to use the function `WFile_nextRecord` which will take as parameter the pointer to the corresponding WFile object. This function will extract the header of the WARC Record pointed by the WFile object, then this last one will jump to the next WARC Record. If safetly used, the file descriptor inside the WFile object will allways point to a valid WARC Record in the WARC file or to the end of file. Hence, we are sure tha we will never badly extract the header of the wanted WARC Rrcord using the `WFile_nextRecord`.

When succeeding in the reading operation, the function `WFile_nextRecord` will return a pointer to a valid WRecord object (the construction of the object is done inside the `WFile_nextRecord`, and then it must be destroyed after usage) which may be manipulated by the WRecord class functions (described in the following sections). If it fails, it will return a NIL (NULL) value in all crash case.

With the `WFile_nextRecord`, we can only read the WARC Record in a fifo order. If we read a WARC Record using it, we can not come back to directly. And when th file descrciptor of the WFile object reaches the end of th WARC file, the `WFile_nextRecord` function will return a NULL value considering it as an error. However, the library provides a function called `WFile_hasMoreRecords` that tests if it remains some WARC Records that can be read. Generally, we use the function `WFile_nextRecord` in collaboration with the `WFile_hasMoreRecord` function like shown in the example below. And if we want to come back to an already visited record, the library provides a function `WFile_seek` that allows to seek an offset in the correspondin WARC file. Generally, the given offset have to be a valid WARC Record offset, else the reading operation will success (when reading from a bad offset, the `WFile_nextRecord` will consider that it is on a corrupted WARC Record).

Example 4-3. Reading safetly the WARC Records of a WARC file opened with the WFile object `warcfile`

```
void *  warcfile =  bless (WFile, "file.warc",0,
          WARC_FILE_READER, WARC_FILE_DETECT_COMPRESSION, ".");
void * r = NIL;

while (WFile_hasMoreRecord (warcfile))
 {
 r = WFile_nextRecord (warcfile);

 unless (r) /* testing if the read operation has passed well*/
 /* some instructions */

 /*

 working with the record r

 */

 destroy (r); /* destruction of the generated object */
 }
```

```
/* end of program */
destroy (warcfile) /* destruction of the object warcfile */
```

4.4.2. The registration of an extracted WARC Record in a WARC file

When a WRecord object is created by the `WFile_nextRecord`, the first thing that we may want to do with it is to recover the corresponding stored data block, and of course the library provides the necessary function to do this (see the section concerning the WRecord routines). But before we can access the data block of a WRecored object, we must register this last one to its origin WFile object. This will oblige the WRecord to be linked with its origin WFile (we may use the same WRecord object with several WFile objects) and we will never have the possiblity to use it to try thye extraction of data from another WARC file. The function which allows us to do that is `WFile_register`. This function will have four parameters which are, in that order:

—The pointer to the corresponding WFile object.

—The pointer to WRecord object we want to register.

—The pointer to a callback function. This parameter is the most important because it is a function given by the user where he will write the code of the work he wants to perform with the data extracted from the WARC Record. This function will be automaticallay called by the WRecord when the data is reclaimed by the user. Hence, the user will never have directly an acces to the WARC file, it is the WFile object itself which will read the data from it gives it the user. The parameters of this callback function will be described in the WRecord data acces routine function called `WRecord_getContent`.

—The pointer to a user environnement structure. Porbably, the user will need the data of a WARC Record to fill some variables and structures. For this, he have to groupe them in a single structur and give the pointer of this last one as a parameter of the `WFile_register`. When the callback function is called, this pointer is given as a parameter to it. Actually, the type of this pointer is void *. It is for the user to do a casting (type change) on it to the wanted structure pointer type in such way that he will be able to use it.

4.4.3. Adding a WARC Record to a WARC file

sect2 Now, if we want to archive a data in a WARC file, we simply have to fill a WRecord object with its parameters (see the WRecord filling section) and call the function `WFile_storeRecord`. This function take two parameters:

—The pointer to the WFile object linked to the WARC file where we want to store the record.

—The pointer to the WRecord object which was filled with the new WARC Record parameters and linked to the data file we want to archive.

Before the new WARC Record is stored, the function `WFile_storeRecord` will check if its fields are filled in conformity with the WARC Format. Then, the WARC Record will only be stored if all its mandatory fields are filled and if the other fields matches the WARC-Type of the WARC Record. If there is an additionnal or missing field w.r.t the WARC-Type, the storage operation is aborted and the `WFile_storRecord` function will return an error.

4.5. WRecord object routines

Now, we will describe the routines provided with the WRecord class. These functions will allow us to set up and recover the main fields of the WRecord object and also its extra fields (see the ISO WARC format specification). There is also functions that gives information about an extracted WARC Record linked to the WRecord object.

4.5.1. WARC Record Fileds getting functions

In the folowing, we give the description of the functions of the WRecord class that allow to recover the fields of a WARC Record. These function are very useful when we want to get the fileds of the header of an extracted WARC Record. For each predefinded WARC Record header field, ther is its corresponding getting function. There is also a mean to recover the Extra non predefined fields (the ANVL fields of the header), then, we may get all the stored information about the WARC Record.

—`WRecord_getWarcId`: This function is used to get the WARC ID of the WARC Record. Remember that the format of a WARC ID is: WARC/version, where version is the actual version of the WARC Record, for example, the lastest version of the WARC is 0.17, then the WARC ID of any WARC Record is `"WARC/0.17"`.

—`WRecord_getRecordType`: This function is used to get the WARC-Type field of the WARC Record. Lets remember that there is eight predefined types for a WARC Record, then, an enumertive type has been used to represent each of theses type. This function will return us the corresponding enumaration to the WARC Type of our WARC Record. The list of possible enumearations is given ordered below:

- `WARC_UNKNOWN_RECORD`: For an eventual unknown WARC Record type.
- `WARC_INFO_RECORD`: For the `"warcinfo"` type.
- `WARC_RESPONSE_RECORD`: For the `"response"` type.
- `WARC_REQUEST_RECORD`: For the `"request"` type.
- `WARC_METADATA_RECORD`: For the `"metadata"` type.
- `WARC_REVISIT_RECORD`: For the `"revisit"` type.
- `WARC_CONVERSION_RECORD`: For the `"conversion"` type.
- `WARC_CONTINUATION_RECORD`: For the `"continuation"` type.
- `WARC_RESOURCE_RECORD`: For the `"resource"` type.

All these enumerations are defined in the public header `"wrectype.h"` that must be included to have the possibility to use them. And if we may add a new WARC Type to this list, we can simply add this new type to the enumartions.

—`WRecord_getRecordId`: This function gives the Id of the WARC Record stored in the WARC-Record-ID field.

—`WRecord_getDate`: This function gives the Creation date of the WARC Record stored in its WARC-Date field.

—`WRecord_getContentLength`: This function gives the size of the data block of the WARC Record. It is stored in the Content-Length field.

—`WRecord_getContentType`: This function gives the type of the content of the WARC Record data block. It is stored in the Content-Type field.

—`WRecord_getConcurrentTo`: This function gives the Id of the WARC Record which the current one is concurrent to. It is stored in the WARC-Concurrent-To field.

—`WRecord_getBlockDigest`: This function gives the digest value of the data block folowing a specific algrithm. It is stored in the WARC-Block-Digest field of the WARC Record header.

—`WRecord_getPayloadDigest`: This function gives the digest value of the data block payload following a specific algorithm. It is stored in the header field WARC-Payload-Digest.

—`WRecord_getIpAddress`: This function gives the Ip address from where the data block was crawled. It is stored in the WARC-IP-Address field.

—`WRecord_getRefersTo`: This function gives the Id of the WARC Record to which the current one refers. It is stored in the field WARC-Refers-To

—`WRecord_getTargetUri`: This function gives the URI of the source from where the data block was crawled. It is stored in the WARC-Target-URI field

—`WRecord_getTruncated`: If the WARC Record has been trucncated, this function will give us the reason of the tructcation

—`WRecord_getWarcInfoId`: This function gives the Id of the `"warcinfo"` Record describing the WARC file to which the current WARC Record belongs. It is stored in the WARC-Warcinfo-Id field.

—`WRecord_getFilename`: This function gives the name of the WARC file to which this WARC Record belongs. It is stored in the WARC-Filename field.

—`WRecord_getProfile`: This function is only useful in the cas of a `"revisit"` Recrod because that this information may only exist in this case. It returns the kind of transformation applied on an already visited data but found under an other form. This value is stored in the WARC-Profile field

—`WRecord_getIdentifiedPayloadType`: This function returns the identified type of the payload of the WARC Record data block. This value is stored in the WARC-Identified-Payload-Type field.

—`WRecord_getSegmentOriginId`: This function is only usable in the case of a fragment of a segmented record identified by the `"continuation"` type. It gives the Id of the first fragment of the whole WARC Record, this one is stored in the WARC-Segment-Origin-Id field

—`WRecord_getSegmentNumber`: This function is also usable in the WARC Record fragment case. It gives the number of current fragment, this value is stored in the WARC-Segment-Number field

—`WRecord_getSegTotalLength`: This function is only usable in with the last fragment of a segmented WARC Record. It gives the total Length of the whole WARC Record knowing that the Content-Length field will only give the actual Length of the data fragment stored in the record. It is stored in the WARC-Segment-Total-Length field.

—`WRecord_getAnvlField`: We know that in the header of a WARC Record, we can find predefined fields described below, and some extra fileds defined by the user himself (if he needs them to add more information to the header). Then, this function offers a mean to recover these user-defined fields. It takes as parameter the pointer of the WRecord object, the rank of the extra field in the header and two strings that will contain both of the key and the value of the extra field.

—`WRecord_getAnvlFieldsNumber`: This function return the number of the extra fields in the WARC Record. It may be useful when combined with the WRecord_getAnvlField.

4.5.2. WARC Record fileds setting function

Now, if we want to store a data block in a AWRC file, we have to crate a WARC Record for it, then, we must fill the headers field fields before. The routines described below allow us to fill each of a WARC Record header fields.

—`WRecord_setRecordType`: This function is used to set the WARC-Type field of the WARC Record. It takes as parameters the pointer of the WREcord object corresponding to trhe WARC Record and one of the constant values described above indicating the WARC Record type. This function returns `False` if it succeeds and True otherwise.

—`WRecord_setRecordId`: This function sets value of the WARC Record Id which will be stored in the WARC-Record-ID field. It takes as parameters the pointer to the corresponding WRecord object and the value of the WARC Record Id. This function will also check if the value of the WARC Record Id matches the ISO URI format. It returns `False` in success case and `True` else.

—`WRecord_setDate`: This function sets the Creation date of the WARC Record which will be stored in its WARC-Date field. It takes as parameters the pointer to the corresponding WRecord object, the value of the creation date field string and its length. The date string format must be like the `"YYY-MM-DDThh:mm:ssZ"` format. It will return `False` if it succeeds and `True` else.

—`WRecord_setContentType`: This function allows to set the value of the Content-Type field of the WARC Record. It will have as parameters the pointer to the WRecord object, the mime-type string describing the type of the data block to be stored and its length. It returns `False` when it succeeds and `True` otherwise.

—`WRecord_setConcurrentTo`: This function is used to set the value of the WARC-Concurrent-To Field of the WARC Record Header. It takes as parameters the pinter to the WRecord Object, the uri string giving the Id of the concurrented WARC Record and its length. When it succeerds, this function returns `False` and `True` otherwise.

—`WRecord_setBlockDigest`: This function sets the value of the WARC-Block-Digest field. it should have as parameters the pointer to the WRecord object, the digest value string and its length. the return value of this function is `False` in success case and `True` otherwise.

—`WRecord_setPayloadDigest`: It allows to set the value of the WARC-Payload-Digest field. It takes the pointer of the WRecord object as a first parameter, the digest string describing the value of the field and its length. It returns the same values as the previous functions.

—`WRecord_setIpAddress`: This function sets the value of the WARC-IP-Address field. It will have the pointer of the WRecord object, the Ip address string as parameters and its length. The retun values of this function are the same as the previous ones.

—`WRecord_setRefersTo`: This function sets the WARC-Refers-To field value of a WARC Record. It has as parameters the pointer to the WRecord object, the uri string giving the refered WARC Record and its length. It returns the same possible values as the previous functions.

—`WRecord_setTargetUri`: It sets the value of the WARC-Target-URI field of a WARC Record. We give it the pinter of the WRecord object, the uri string of the targeted WARC Record and its length. This function also returns `False` in success case and `True` otherwise.

—`WRecord_setTruncated`: Sets the WARC-Truncated field value of a WRecord object, identified by its pointer given in the first parameter, with the truncation reason string, given in the second parameter, and its length in the third parameter. It returns the same values as the prvious functions.

—`WRecord_setWarcInfoId`: It sets the WARC-Info-ID field of a WARC Record. It takes as parameters the pinter to the WRecord object, the uri string describing the id of the related WARC-INFO Record and its length. It also returns `False` whend it succeeds and `True` otherwise.

—`WRecord_setFilename`: This function sets the WARC-Filename field of a WARC Record. it has as parameters the pointer to the WRecord object, the WARC file name string and its length. When it succeeds, this function returns `False` and it returns `True` otherwise.

—`WRecord_setProfile`: This function is only useable in the cas of a `"revisit"` Recrod. It sets WARC-Profile field of a WARC Record having the pointer to the WRecord object, the porfile string as parameters and its length. It returns `False` if it succeeds and `True` otherwise.

—`WRecord_setIdentifiedPayloadType`: It sets the WARC-Identified-Payload-Type field of the WARC Record. It will take as parameters the pointer to the WRecord object, the mime-type string describing the payload type and its length. It has the same return values as the previous functions.

—`WRecord_setSegmentOriginId`: This function is only usable in the case of a fragment of a segmented record identified by the `"continuation"` type. It stets the value of the WARC-Segment-Origin-ID field of the WARC Record. It takes the pointer to the WRecord object, the id string and its length as parameters. Its returns values are the same as the previous functions.

—`WRecord_setSegmentNumber`: This function is also usable in the WARC Record fragment case. It sets the WARC-Segment-Number field of the WARC Record. Its parameters are the pointer to the WRecord object and the integer giving the number of the segment. it returns the same thing as the previous functions.

—`WRecord_setSegTotalLength`: This function is only usable in with the last fragment of a segmented WARC Record. It sets the WARC-Segmenet-Total-Length field of this particular WARC Record. we give it as parameters the pointer to the WRecord object and the total length as an integer. It will return `False` if it succeeds, and `True` otherwise.

—`WRecord_addAnvl`: This function is used to add an unprdefined field at the end of the header of a WARC Record. It will take as parameters the pointer to the WRecord object, the field key string and its length, and finally the field value string and its length. The return values of this function are also the same as the prvious setting functions.

4.5.3. WARC Record data block recovering

To recover the data stored in a WARC Record, we will never allow to the user to directly access the WARC file. A content recovering function has been made to allow the user to manipulate the data block contained in the WARC Record. This function is `WRecord_getContent`. It may onlye be used after having registred the WRecord object using the `WFile_register` function. This function will call the registred callback function using on the data of the WARC Record.

4.5.3.1. Tha callback function prototype

The callback function is e function having a particular prototype where the user will give the processing to do with the data coming from the WARC Record data block. The prottype of a callback function is:

Example 4-4. WARC callback functions prototype

```
warc_bool_t callback (void * env, const char* buff, const warc_u32_t size)
```

Where `env` is the pointer to the user envronnement that will be filled during the running of the callback function. The user has to do a cast inside the callback function to the `env` variable in such way that it will be usable. The `buff` parameter is a pointer to a byte array that will hold a chunk of data from the WARC Record data block and the parameter `size` will give the size of the data held in `buff`.

The `WRecord_getContent` will take a chunk of `size` bytes from the data block of the WARC Record, puts it in an array buffer `buff` and will give them with the user environnement `env` as the parameters of the user callback function it will call on. This operation is repeated until all the data block will be all given. Thus, the user must take care that the callback function will recieve the data block by chunks, and it has to deal with that during the processing. The user is free to apply any processing on the data that the callback function will recieve. And he may also give a `NULL` pointer as environnement parameter if he has no one.

4.5.4. Other useful WRecord routines

There is three other WRecord objects routines that are not directly concerned by the WARC format. The first function is `WRecord_getOffset` returns the offset of the WARC Record in the WARC file. The two other functions are `WRecord_getCompressedSize` and `WRecord_getUncompressedSize` are useful in the case of a compressed WARC Record. They gives respectively the compressed and the uncompressed size of the WARC Record. In the case of an Uncompressed Record, this previous values are the same. All these functions takes as parameter the pointer to the WRecord object

4.6. ARC files manipulation routines

The warc-tools library offers also function to manipulate archives stored in file with the old ARC format. As for the WARC format, there is two categories of functions: some functions are used in the manipulation of the ARC files and the others to manipulate the ARC Records.

4.6.1. Afile object routines

In this section, we will present the functions that allows the manipulation of the ARC files. But, because that the warc-tools library is not dedicated to the ARC format, there will not be functions to store data in ARC files. There is only functions to recover data from the ARC file. the main purpuse of adding ARC file manipulation routines is to have the possibily to convert whole ARC files to WARC files.

- `AFile_nextRecord` This function allows to get an ARC Record from the ARC file. It has the same comportment as the `WFile_nextRecord` function. It allows to sequentially recover the ARC Records of an ARC file until it reches the end of the file. This function takes as parameter the pointer to the AFile object. It returns the pointer to a valid `ARecord` object if it succeeds or `NULL` otherwise.
- `AFile_hasMoreRecords` This function allows to test if there is more ARC Records remaining in the ARC file. It takes as parameter the pointer to the AFile object. It reurns `True` if still there is more ARC Records to explore in the ARC file or `False` if we are at the end of the ARC file.
- `AFile_seek` This function is used to seek a particular offset in the ARC file. of course, if we seek an offset which is not the begibing of a valid ARC Record, the function `AFile_nextRecord` will return `NULL`. It takes as parameter the pointer of the AFile object. It will return `False` when it succeeds and `True` otherwise.
- `AFile_register` It has the same role as the WFile routine `WFile_register`. It allows to register an extracted ARecord object with the corresponding ARC file in the purpuse of extracting the data block stored inside. It must have as parameters the pointer of the AFile object, the pointer of the ARecord objec, the pointer of the callback function used to manipulate the ARC Record data and finally the pointer of the user environnement variable. This function will return `False` if the registration succeeds and `False` otherwise.

4.6.2. ARecord object routines

The following functions are used to recover the hedaer field of an ARC Record and also its data block. As we said previously, our aim is not to create ARC files, but only to recover information from them. Then, there will be no sttong function for an ARecord object.

- `ARecord_getUrl` This function is used to get the URL field of an ARC Record. It takes as parameter the pointer to the ARC Record object. It returns the URL string when it succeeds or `NULL` otherwise.
- `ARecord_getreationDate` This function gives the creation date field of the ARC Record. It will have the pointer of the ARecord object as parameter. It returns the date string if it succeeds and `NULL` otherwise.

- `ARecord_getMimeType` This function is used to get the mime type field of an ARC Record. It takes the pointer of the ARecord object as parameters and it returns the mime type string in the success case or `NULL` otherwise.
- `ARecord_getIpAddress` This function will return the ip address field of the ARC Record header. it takes as parameter the pinter to the ARecord object. It will return the string describing the ip address filed if it succeeds or `NULL` otherwise.

4.6.3. Conversion of an ARC Record to a WARC Record

Our aim to offer routines which allow the extraction of records from ARC files is to convert these last one into WARC Records. The, after having extracted an ARC Record rom an ARC File, we may create a new WRecord object and fill its hederd fields with nex information or inspired from the hedaer of the ARC Record. However, we must note two things:

—The format of the ARC dates ins different from the one of the WARC dates. The fisrt follows the standard fromat `"YYYYMMDDHHMMSS"` but the second follows a specila WARC format which is `"YYYY-MM-DDTHH:MM:SSZ"`. Then it is clear that we can not directly put the date extracted from an ARC Record in the WARC Record fields or the setting function will return an error. To this purpuse, a function called `WRecord_setDateFromArc` is offered. It takes as parameters the pointer of the WRecord object, the string describing the standard formatted date and the its length. this function will convert the standard date into a WARC formatted date and fill the WARC-Date field of the WRecord object. It will return `False` if it succeeds and `True` otherwise.

—To facilitate the tranformation of an ARC Record to a WARC Record, the warc-tools library offers a function called `ARecord_transferContent`. As its name indicates, this functions allows to transfer the data block from the ARC Record to the newly created WARC Record. It takes as parameter the pointer of the ARrecord object, the pointer to the new WRecord object and the pointer of the WFile object from where the data will be tranfered. Take care to give the right AFile object from which the ARecord object has been extraced because that the `ARecord_transferContent` function exÃ©cute a registration operation of the ARecord object with the AFile object. Then, if the two objects are not linked together, the transfer wilol fail. This function returns `False` if it succeeds and `True` otherwise.