# bwtdisk

A program and library for computing the BWT of very large files



Version 0.9.0 of 2 February 2010

**Giovanni Manzini**

This file documents the `bwtdisk` library version 0.9.0, written by Giovanni Manzini (giovanni.manzini@unipmn.it).

# Table of Contents

# 1 Introduction

The Burrows-Wheeler Transform (BWT from now on) is a data transformation whose main applications are Data Compression and the construction of full text indices. During the last few years many tools have been developed for the computation of the BWT: for most files the best choice is probably the `libdivsufsort` library by Yuta Mori (available at http://code.google.com/p/libdivsufsort/). Unfortunately `libdivsufsort` cannot be used for very large input files: the input size $n$ must be less than $2^{31}$, and to work efficiently it requires $5n$ bytes of free RAM.

If you have to work with huge files, or you are allowed to use only a limited amount of RAM, `libdivsufsort` must be ruled out and the `bwtdisk` library comes to the rescue! `bwtdisk` can work with input files of size up to $2^{63}$ and can use a preassigned amount of RAM: of course the larger the amount of available RAM the faster is the computation. This is possible thanks to an appropriate use of external memory. `bwtdisk` works by building larger and larger BWT's: the input and the partial (and final) BWT's are stored on disk in such a way that at any given moment only the input and at most two partial BWT are stored on disk. Hence `bwtdisk` uses the amount of RAM specified by the user and up to $3n$ bytes of disk space.

A fundamental feature of `bwtdisk` is that all files on disk are accessed only by sequential scans, thus it takes full advantage of modern disk features that make sequential disk accesses much faster than random accesses. Another important advantage of the sequential disk access is that all files (input, output, and intermediate) can be stored in compressed form, thus further reducing the disk usage and the total I/Os. The major limitation of the `bwtdisk` library is that it works using one byte per character, so it can only handle alphabets of size at most 256. This limitation is shared by `libdivsufsort` and, to our knowledge, by all other publicly available tools for computing the BWT.

For efficiency reasons the `bwtdisk` library computes the BWT of the input file *reversed*. For most applications this is not a problem[1]; if one really needs the BWT of the unreversed input it suffices to use the `text_rev` tool to reverse the input text before the BWT computation: See Section 3.3 [text_rev], page 9.

For a complete description of the algorithms underlying the `bwtdisk` library see *Lightweight Data Indexing and Compression in External Memory* by P. Ferragina, T. Gagie, G. Manzini, Proceeding LATIN 2010, Springer Verlag Lecture Notes in Computer Science (a preliminary version is available at http://arxiv.org/abs/0909.4341).

[1] Indeed, the BWT of the input $T$ reversed, call it BWT($T^R$), has some advantages with respect to BWT($T^R$). For example, the standard BWT inversion algorithm produces the input $T$ reversed: if we apply the standard algorithm to BWT($T^R$) it will produce the original input $T$.

# 2 The bwte command line tool

After the successful compilation of the library the `bwte` tool can be used to compute the BWT from the command line.

## 2.1 Basic Usage

The simplest (but not the best!) usage of the `bwte` tool is the following:

```
bwte infile -m imem
```

This command computes the BWT of the file named *infile* using at most *imem* megabytes of RAM (plus a constant overhead of a few megabytes). Recall that `bwte` computes the BWT of the input file *reversed* (see Section 3.3 [text_rev], page 9).

`bwte` stores its output in two files: *infile*`.bwt` containing the BWT proper, and *infile*`.bwt.aux` containing the number of occurrences of each character in the input file. This last piece of information is useful for inverting the BWT or for building compressed indices: it can be safely deleted if one is only interested in the BWT.

The format of the output files is described in the next section. Note that the above simple command produces an *uncompressed* BWT. This is not recommended for several reasons: see Section 2.3 [Using filters], page 2 for how to produce a compressed BWT.

## 2.2 Output files format

Assuming that the input file has size $n$, the *uncompressed* BWT file produced by `bwte` (default extension `.bwt`) has the following format:

| | |
|---|---|
| 8 bytes | The integer $n+1$ in little-endian format |
| 8 bytes | Position $p$ of the EOF symbol in the BWT in little endian format |
| $n+1$ bytes | The BWT proper |

The uncompressed BWT file has size exactly $n+17$ bytes. The value $p$, $0 \leq p \leq n+1$, is the position in the BWT of the special EOF (end-of-file) symbol . No such special symbol is actually stored in the BWT: the $p$-th entry of the BWT contains an arbitrary character that must be ignored since it is not really part of the BWT.

The second file produced by the `bwte` tool has default extension `.bwt.aux` and contains 256 `int64_t` integers, in little-endian format, giving the number of occurrences of characters `0`, `1`, . . . , `255` in the input file. The file size is therefore exactly 2048 bytes. Even when the BWT file is stored in compressed form this auxiliary file is never compressed. If necessary, this file can be generated using the `text_count` tool. See Section 3.2 [text_count], page 8.

## 2.3 Using filters

Files, especially when large, are usually given to us in a compressed format produced using either standard tools (like `gzip`) or ad-hoc tools (think for example to a compressor for genome sequences using two bits per base). `bwte` has the possibility of working with a compressed input file. This feature has a twofold advantage:

1. No extra disk space is required to store the uncompressed file;

2. Fewer I/Os are required to read a compressed file: in many cases the reduction in the I/O volume more than compensate the extra CPU time required for decompression. As a result, `bwte` often runs faster when the input is compressed.

The same considerations apply for the BWT file. Indeed, the BWT is a transformation designed to make the data more compressible hence working with a compressed BWT is likely to significantly reduce the running time.[1] For these reasons, the possibility of working with compressed input and output files is a fundamental feature of `bwte` and of the `bwtdisk` library in general.

### 2.3.1 Predefined filters

The following filters are available in any version of `bwte`/`bwtdisk`.

| *Filter ID* | *Extension* | *Description* |
|---|---|---|
| 0 | none | no compression |
| 1 | `.gz` | gzip compression |
| 2 | `.rrc` | rle + range encoding |
| 3 | `.atn` | DNA 5 symbols (ACGTN) encoding |
| 4 | `.lzma` | lzma compression |

Note that each filter can be used both as an input filter (that is, to handle a compressed input file) and as an output filter (that is, to produce a compressed BWT file).

More filters will be hopefully included in the future. To see a list of the available filters, and their associated *ID* and *Extension*, type `bwte` on the command line without any argument. A brief description of the pre-defined filters is given below. With a little experimenting you should be able to find the filter more suitable for your application: which is the best filter usually depends on the interplay between data compressibility, cpu speed, and disk transfer speed.

#### Filter 0: no compression

This is the default filter and stores the data without any modification. To be used when you do not want to meddle with compressed files.

#### Filter 1: gzip compression

This filter compresses data using the `zlib` library (see http://www.zlib.net/). It is very useful as an input filter since data is often stored in gzip format. It does a fair job also as an output filter, but there could be better alternatives.

#### Filter 2: rle + range encoding

This is an especially designed compressor which is one of the best for compressing BWT files; it is recommended as a general purpose output filter.

#### Filter 3: DNA 5 symbols encoding

This is an especially designed compressor for sequences over the alphabet {A, C, G, T, N} (it will fail if the input contains any other symbol). It encodes runs of N's using run length

[1] In our experiments, an 8GB collection of html pages is stored in 1.7GB when compressed with gzip. The compressed BWT of the same collection takes only 440MB!

encoding, and uses 2 bits for each one of the four bases A, C, G, T. It should be used as an input and output filter for genomic sequences.

### Filter 4: lzma compression

The `lzma` compressor (see http://www.7-zip.org/) is a somewhat recent evolution of `gzip` and is one of the most effective general purpose compressors. Since `lzma`-compressed files are becoming common this is an useful input filter. It is not recommend to use this as an output filter since writing in `lzma` format is implemented using a, rather inefficient, pipe mechanism (which will probably not work at all in non-Unix systems).

## 2.3.2 Choosing a filter

To use an input and/or output filter, the `bwte` tool must be invoked with the following syntax (as usual arguments in brackets are optional):

```
bwte infile -m mem [-o outfile] [-t ifil] [-b ofil]
```

If the parameter *ifil* is present and non-negative then the filter with ID equals to *ifil* is used as input filter. If *ifil* is negative, `bwte` tries to autodetect the input filter: if the extension of one of the known filters is a suffix of *infile* that filter is used, otherwise filter 0 (no compression) is used. Filter autodetection is the default behavior when the `-t` option is not present.

Similarly, the parameter *ofil* can be used to specify the output filter. If *ofil* is present and non-negative then the filter with ID equals to *ofil* is used as output filter. If *ofil* is negative, the *outfile* parameter must be given, and `bwte` will try to autodetect the output filter as above. The default behavior if the `-b` option is not present is to use filter 0 (no compression).

Note that if *ofil* is given and *outfile* is not, then the name of the output file will be *infile*`.bwt.ext` where `.ext` is the extension associated to filter *ofil*.

## 2.3.3 Defining new filters

It is possible to define new filters in order to provide ad-hoc compression for particular types of data. This feature requires a good knowledge of C programming since, not only you have to write your own code, but you also have to make sure that your code is compiled and linked with the rest of the `bwtdisk` library. The following step by step example should guide you in this task. See also the file `filters/plain_filter.c` which contains the code for the plain filter (the one that simply copy data without compressing it).

### 2.3.3.1 Required functions

Suppose you have devised an extremely efficient compression format, let us call it the `zxy` format. To use it inside the `bwtdisk` library you have to write functions for reading and writing in that format. Fortunately, since `bwtdisk` library reads and writes data only by sequential scans, you only need functions for reading/writing the next $n$ bytes: there is no need to implement complex seek operations in the compressed data.

Using your favorite editor, create the file `zxy_filter.c` in the directory `filters`. Inside it add the statement `#include "filters.h"` as well as any other `include` needed by your code. Then add the code for the following functions.

### `void *_bwtext_zxy_open(char *filename, int32_t mode)`

If *mode* is `'r'` opens the file named *filename* for reading, if *mode* is `'w'` opens *filename* for writing. Every other value for *mode* is illegal. Returns a pointer that is later passed to the functions that do the actual reading or writing. Returns `NULL` on errors.

### `int32_t _bwtext_zxy_read(void *f, uint8_t *buf, int32_t n)`

Reads up to *n* bytes and stores them to *buf* that must be a previously allocated buffer. *f* must be a pointer previously returned from a call to `_bwtext_zxy_open` with mode equals `'r'`. Returns the number of bytes actually read that can be less than *n* if the end of file is encountered.

### `int32_t _bwtext_zxy_write(void *f, uint8_t *buf, int32_t n)`

Reads *n* bytes from *buf* and writes them to the file associated to *f*. *f* must be a pointer previously returned from a call to `_bwtext_zxy_open` with mode equals `'w'`. Returns the number of bytes actually written; it is generally an error if this number is less than *n* (it means there is no space on disk or some other error occurred).

### `void _bwtext_zxy_rewind(void *f)`

*f* must be a pointer previously returned from a call to `_bwtext_zxy_open` with mode equals `'r'`. This function rewinds the file associated *f* so that the next calls to `_bwtext_zxy_read` fetch data from the beginning of the file.

### `int64_t _bwtext_zxy_close(void *f)`

Closes the file associated to *f* which must be a pointer previously returned from a call to `_bwtext_zxy_open`. After this call no more read/write operations are possible on *f*. This function should free any memory allocated by `_bwtext_zxy_open`. If possible, the function should return the number of bytes actually read/written from disk (after compression). This value is only used for debugging/informative purposes, so if it is not available it is safe to return 0. Returns -1 on errors.

### `void _bwtext_register_zxy_filter(filter *fil)`

Stores in the filter data structure *fil* the name of the functions defined above and some auxiliary information. In our example the code of this function should be something like

```
void _bwtext_register_zxy_filter(filter *fil) {
  fil->open  = _bwtext_zxy_open;
  fil->read  = _bwtext_zxy_read;
  fil->write = _bwtext_zxy_write;
  fil->rewind = _bwtext_zxy_rewind;
  fil->close  = _bwtext_zxy_close;
  fil->name   = "zxy super-compression";
  fil->extension = ".zxy";
}
```

The `name` field is used to describe the filter for example when the command `bwte` is given without arguments. The `extension` field is used for example for filter autodetection (see Section 2.3.2 [Choosing a filter], page 4) and must be different from the extension used by all other filters.

#### 2.3.3.2 Filter parameters

Using both `bwte` and the `bwtdisk` library, it is possible to pass to an output filter two parameters that can influence how the compressor operates. For example, in the rle + range coding filter these parameters are used to control how quickly range coding "adapts" to new statistics. `bwte` and `bwtdisk` store these parameters in the global variables `_bwtext_FilterPar1` and `_bwtext_FilterPar2` which are of type `double`. If a significant value is stored in one or both of these variables, a nonzero value is stored in the corresponding variable `_bwtext_FilterPar1_flag` or `_bwtext_FilterPar2_flag` that otherwise contains zero. If a filter does use `_bwtext_FilterPar1` and `_bwtext_FilterPar2` it has to provide default values for them in case one or both `_bwtext_FilterPar1_flag` or `_bwtext_FilterPar2_flag` are zero (which means that `bwte`/`bwtdisk` did not provide the corresponding parameter).

Note that the `_bwtext_FilterPar1_flag` and `_bwtext_FilterPar2_flag` can be specified only when a filter is used as an output filter, that is, to compress data. These parameters are never specified when a filter is used as an input filter, that is, to decompress data. This means, that the parameters used for compression must be stored in the compressed file so that decompression can be done without any external help.

#### 2.3.3.3 Putting all together

When the code for your filter is complete, you must let `bwtdisk` know about it. In the file `filters/filters.c` increase by one the constant `NUM_TEXT_FILTERS`, and add the prototype for `_bwtext_register_zxy_filter`. Then, in the function `_bwtext_register_text_filters` add the statement

```
_bwtext_register_zxy_filter(&tf[i++]);
```

immediately after the similar calls to the other `_bwtext_register_*` functions.

Finally, in the file `filters/makefile` add `zxy_filter.o` at the end of the definition of the `OBJECTS` variable. Next time you rebuild the library your filter should be there: the command `bwte` with no arguments should list also its name and associated extension.

## 2.4 bwte man page

This section contains a sort of man page of the `bwte` command line tool. Note that some of the concepts are explained in the previous sections. Arguments in brackets are optional.

### Synopsis

```
bwte [-v][-t fil][-b fil][-m imem][-o bwtfile][-y val][-z val]  infile
```

### Description

Computes the Burrows-Wheeler transform of the file *infile reversed* using a preassigned amount of RAM and external memory if necessary. Input, output and intermediary files can be stored in compressed form using an extendible filter mechanism.

### Options

`-m imem`

Amount of RAM, in megabytes, that will be used by `bwte`. Default is 256, but if you have more you should use it since more RAM significantly reduces

running time. In addition to the amount specified by this option `bwte` will use only a few extra megabytes; note that RAM usage is not affected by the size of the input file.

`-t fil`

Specify the filter to be used to read the input file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the input file name that filter is used. If no extension matches filter 0 (no compression) is used. The default for this option is filter autodetection. See Section 2.3 [Using filters], page 2.

`-b fil`

Specify the filter to be used to write the BWT file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the output file name that filter is used. If no extension matches filter 0 (no compression) is used. Filter autodetection is possible only if the BWT file name is specified with the `-o` option. The default for this option is filter 0 (no compression). See Section 2.3 [Using filters], page 2.

`-o outfile`

Specify the name of the output file (the one containing the BWT). The default is the input file name followed by the extension `.bwt` followed by the extension associated to the output file filter. In addition to the file containing the BWT, `bwte` produces a file containing the number of occurrences of each character in the input file. The name of this auxiliary file is the name of the BWT file with the additional extension `.aux`. See Section 2.2 [Output files format], page 2.

`-y val -z val`

Specify two floating point values that are stored in two global variables for later use by the BWT file filter. See Section 2.3.3.2 [Filter parameters], page 6.

`-v`

Verbose mode: Show information on the progress of the computation. Further `-v`'s increase the verbosity level.

# 3 The bwtdisk tools

The successful compilation of the `bwtdisk` library produces, in addition to `bwte` (see Chapter 2 [The bwte command line tool], page 2), some other useful command line tools.

## 3.1 text_conv

`text_conv` converts files among compressed formats using the filters defined in the `bwte`/`bwtdisk` library.

### Synopsis

```
text_conv [-i fil][-o fil][-y val][-z val][-v]  infile  outfile
```

### Description

Convert between different compression formats by reading the content of *infile* using a given filter, and writing it to *outfile* using a different filter.

### Options

`-i fil`

Specify the filter to be used to read the input file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the input file name that filter is used. If no extension matches filter 0 (no compression) is used. The default for this option is filter autodetection.

`-o fil`

Specify the filter to be used to write the output file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the output file name that filter is used. If no extension matches filter 0 (no compression) is used. The default for this option is filter 0 (no compression).

`-y val -z val`

Specify two floating point values that are passed as parameters to the output file filter. See Section 2.3.3.2 [Filter parameters], page 6.

`-v`

Verbose mode: Write to `stderr` information on the progress of the computation. Further `-v`'s increase the verbosity level.

## 3.2 text_count

`text_count` generates the auxiliary file containing the number of occurrences of each character in a given file. The auxiliary file therefore contains 256 `uint64_t` integers in little-endian format and is usually generated by the `bwte` tool. See Section 2.2 [Output files format], page 2. This auxiliary file can be useful for inverting the BWT or building compressed indexes. It is used for building a compressed index with the FM64 library that requires that this auxiliary file exists and has the same name as the BWT file with the additional extension `.aux`. See section "Building the index" in the documentation of the FM64 library.

### Synopsis

```
text_count [-i fil][-v]  infile  outfile
```

### Description

Write to *outfile* the number of occurrences of each character in *infile*.

### Options

`-i fil`

Specify the filter to be used to read the input file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the input file name that filter is used. If no extension matches filter 0 (no compression) is used. The default for this option is filter autodetection.

`-v`

Verbose mode: Write to `stderr` information on the progress of the computation. Further `-v`'s increase the verbosity level.

## 3.3 text_rev

Given an uncompressed file, `text_rev` produces a new file containing the content of the original file *reversed* and possibly compressed. Since `bwte` computes the BWT of the input file reversed, `text_rev` can be used before the invocation of `bwte` to compute the BWT of original un-reversed input.

### Synopsis

```
text_rev [-o fil][-y val][-z val][-v]  infile  outfile
```

### Description

Write to *outfile* the content of *infile* reversed. A filter can be used to compress the content of *outfile*, while the input file *infile* must be in the plain (uncompressed) format.

### Options

`-o fil`

Specify the filter to be used to write the output file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the output file name that filter is used. If no extension matches filter 0 (no compression) is used. The default for this option is filter 0 (no compression).

`-y val -z val`

Specify two floating point values that are passed as parameters to the output file filter. See Section 2.3.3.2 [Filter parameters], page 6.

`-v`

Verbose mode: Write to `stderr` information on the progress of the computation. Further `-v`'s increase the verbosity level.

## 3.4 unbwti

`unbwti` computes the inverse Burrows-Wheeler Transform. It takes as input a BWT file produced by the `bwte` tool and reconstructs the original text. `unbwti` supports compressed formats for both the BWT and the text file using the filters defined in the `bwte/bwtdisk` library. Note that `unbwti` does not require the auxiliary file with the number of occurrence of each character generated by the `bwte` or `text_count` tools.

Since `unbwti` works in internal memory it has some limitations: it uses an amount of RAM equals to four times the size of the original (uncompressed) text and such size must be less that 4 gigabytes. To invert the BWT for larger files you should use the FM64 library.

### Synopsis

```
unbwti [-b fil][-t fil][-o textfile][-y val][-z val][-v]  bwtfile
```

### Description

Invert the BWT stored in *bwtfile* reconstructing the original text. The BWT in *bwtfile* must be in the format produced by `bwte/bwtdisk` (see Section 2.2 [Output files format], page 2) possibly compressed with one of the supported filters.

### Options

`-b fil`

Specify the filter to be used to read the input BWT file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the input file name that filter is used. If no extension matches filter 0 (no compression) is used. The default for this option is filter autodetection.

`-t fil`

Specify the filter to be used to write the text file. A non-negative integer is interpreted as the filter Id. A negative integer indicates that the filter should be autodetected: if the extension of one of the known filters is a suffix of the output file name that filter is used. If no extension matches filter 0 (no compression) is used. The default for this option is filter 0 (no compression).

`-o outfile`

Specify the name of the output file (the one containing the reconstructed text). The default is the name of the BWT file followed by the extension `.twb` followed by the extension associated to the output file filter.

`-y val -z val`

Specify two floating point values that are passed as parameters to the filter used to write the output (text) file. See Section 2.3.3.2 [Filter parameters], page 6.

`-v`

Verbose mode: Write to `stderr` information on the progress of the computation. Further `-v`'s increase the verbosity level.

# 4 The bwtdisk library

The following functions can be used to compute the BWT in external memory from within another application. The first group of functions are responsible for the actual computation of the BWT while the second group provide read access to BWT files. The functions in both groups support the use of filters. See Section 2.3 [Using filters], page 2. To use these functions it is necessary to include the `bwtext_defs.h` file.

## 4.1 BWT computation

### `_bwtext_sufsort`

```
int64_t _bwtext_sufsort( char *tnam,
                         filter *tfil,
                         char *bnam,
                         filter *bfil,
                         int32_t imem);
```

Computes the BWT of the content of file named *tnam* using at most *imem* megabytes of internal memory (plus a constant overhead of a few megabytes). Like the `bwte` tool, this function computes the BWT of the input file *reversed*. The BWT is written to file *bnam*. The variables *tfil* and *bfil* must be pointers to filters that will be used for reading *tnam* and writing *bnam* respectively. These pointers should be initialized with a call to `_bwtext_get_filter` (see below).

The variables `tnam`, `tfil`, and `bfil` must be non-`NULL`. The variable *bnam* can be `NULL`; in this case the output filename is obtained adding to *tnam* the extension `.bwt` followed by the extension associated to filter `bfil`. The function produces also the auxiliary file containing the number of occurrences of each character in the input file (see Section 2.2 [Output files format], page 2). The name of this file is always the name of the BWT file follows by the extension `.aux`.

The caller can initialize the global variables `_bwtext_FilterPar1`, `_bwtext_FilterPar1_flag`, `_bwtext_FilterPar2`, and `_bwtext_FilterPar2_flag` is the filter `bfil` can be controlled using one or both of the parameters `_bwtext_FilterPar1` and `_bwtext_FilterPar2`. See Section 2.3.3.2 [Filter parameters], page 6.

The caller can store a positive integer to the global variable `_bwtext_Verbose` to activate verbose mode: information on the progress of the computation will be then sent to `stderr`. An higher value stored in `_bwtext_Verbose` increases the verbosity level.

### `_bwtext_get_filter`

```
filter *_bwtext_get_filter( int32_t opt,
                            char *filename);
```

Returns a pointer to a filter determined using *opt* and *filename* as follows. If *opt* is non-negative returns a pointer to the filter with ID *opt* (or `NULL` if such filter does not exist). If *opt* is negative the filter is autodetected: if the extension of one of the known filters is a suffix of *filename* that filter is returned, otherwise filter 0 (no compression) is returned. Returns `NULL` if *opt* is negative and *filename* is `NULL`.

### _bwtext_print_available_filters

```
void _bwtext_print_available_filters( FILE *fname);
```

Outputs to file *fname* a human readable description of the available filters together with their associated ID and extensions. This is the same output obtained typing `bwte` with no arguments on the command line.

## 4.2 Access to BWT files

The following functions are designed to provide read access to BWT files produced by the `_bwtext_sufsort` function.

### _bwtext_bwt_open_read

```
bwtfile *_bwtext_bwt_open_read( char *fname,
                                filter *fil);
```

Opens the BWT file named *fname* for reading using the filter *fil*. The file *fname* must be in the proper format, namely the one produced by `_bwtext_sufsort`.

The function returns a pointer to a `bwtfile` struct that can be later used to read the BWT symbols. Note that after the call to `_bwtext_bwt_open_read` the fields `size` and `eofpos` of the returned `bwtfile` structure contain respectively the number of symbols in the BWT, and the position in the BWT of the end-of-file special symbol. See Section 2.2 [Output files format], page 2.

### _bwtext_bwt_read

```
bwtfile *_bwtext_bwt_read( bwtfile *bwt,
                           uint8_t *buf,
                           int32_t num);
```

Reads up to *num* bytes and store them to buffer *buf* that must have been properly allocated. *bwt* must be a pointer previously returned from a call to `_bwtext_bwt_open_read`. Returns the number of actually read bytes that can be less than *num* if the end of the BWT is encountered.

### _bwtext_bwt_close

```
void _bwtext_bwt_close(bwtfile *bwt);
```

Closes the file associated to the structure pointed to by *bwt*. After this call no further operations on the BWT file are possibile.

# Function index