SimpleZip Package

Version 2.1
June 2024

Gray Watson

Table of Contents

SimpleZip

Version 2.1 – June 2024

This package provides Java classes to read and write Zip files. There are a number of different libraries that do this (including one built into the JDK) but I've not found any that gave me precise controls over the Zip internal, persisted data structures. This library allows you to control the output of all Zip data and should allow you to read and write Zip files with full precision.

To get started quickly using SimpleZip, see Chapter 1 [Quick Start], page 2. You can also take a look at the examples section of the document which has various working code packages. See Chapter 4 [Examples], page 10. There is also a HTML version of this documentation. For more information, see the SimpleZip home page.

Gray Watson `http://256stuff.com/gray/`

1 Start Using Quickly

To use SimpleZip you need to do the following. For more information, see Chapter 2 [Using], page 3.

First download SimpleZip from the SimpleZip release page. See Section 2.1 [Downloading], page 3. Or enable via maven. See Section 2.4 [Maven], page 8.

To read Zip files, you use the `ZipFileInput` class. Something like the following where `input` is a `File` or `InputStream`:

```
ZipFileInput zipInput = new ZipFileInput(input);
// readFileHeader() will return null when no more files to read
ZipFileHeader header = zipInput.readFileHeader();
// read file data and write to File (can read to buffer or OutputStream)
zipInput.readFileDataToFile(new File(header.getFileName());
// repeat until readFileHeader() returns null
// optionally read all of the directory entries and set permissions
zipInput.readDirectoryFileHeadersAndAssignPermissions();
zipInput.close();
```

To write Zip files you use the `ZipFileOutput` class. Something like the following where `input` is a `File` or `OutputStream`:

```
ZipFileOutput zipOutput = new ZipFileOutput(output);
// write a file-header to the zip-file
zipOutput.writeFileHeader(
ZipFileHeader.builder().withFileName("hello.txt").build());
// write file data from File (can write buffer or InputStream)
zipOutput.writeFileData(new File("hello.txt"));
// ... repeat until all headers and file-data written
zipOutput.close();
```

For more extensive instructions, see Chapter 2 [Using], page 3.

2 Using SimpleZip

2.1 Downloading Jar

To get started with SimpleZip, you will need to download the jar file. The SimpleZip release page is the default repository but the jars are also available from the central maven repository.

The code works with Java 8 or later.

2.2 Reading Zip Files

2.2.1 Constructing a ZipFileInput

The main class that reads in Zip files is `ZipFileInput`. You can read in Zip data from a file-path string, `File`, or read it from an `InputStream`.

```
// read a file-path
ZipFileInput zipInput = new ZipFileInput("/tmp/file.zip");
// read a file
ZipFileInput zipInput = new ZipFileInput(new File("/tmp/file.zip"));
// read an InputStream
ZipFileInput zipInput = new ZipFileInput(inputStream);
```

2.2.2 Reading Zip File Header Entries

Each file stored in a Zip file is preceded by a header record. You must first read in the header which contains the file-name and other metadata.

```
ZipFileHeader fileHeader = zipInput.readFileHeader();
```

The header contains the following information for each file entry:

- format, version needed to decode
- general-purpose flags
- compression method
- last modified time
- last modified date
- crc32 checksum
- compressed size
- uncompressed size
- file-name bytes
- extra field metadata bytes

The SimpleZip class representing the file-headser is ZipFileHeader.java.

If the crc32, compressed size, or uncompressed size fields are 0 then a data-descriptor will be written after the file-data. See [Data Descriptor], page 4.

Immediately following the file-header is the file-data. If there are no more files to be read then `readFileHeader()` will return null.

2.2.3 Reading File Data to Buffer, File, or Stream

After reading the header, you can then read in the file data. You can have the `ZipFileInput` read the file-data and write the bytes to a file-path string, `File`, or to an `OutputStream`.

```
// read data and write to file output-path, typically from header
zipInput.readFileDataToFile(fileHeader.getFileName());
// or to a file directly
zipInput.readFileDataToFile(new File(fileHeader.getFileName()));
// or to an output stream, such as
ByteArrayOutputStream baos = new ByteArrayOutputStream();
zipInput.readFileData(baos);
```

You can also have `ZipFileInput` read file data as a series of buffers so you can stream large files. You should call `readFileDataPart(...)` until it returns EOF (-1).

```
byte[] buffer = new byte[4096];
while (true) {
   // can also read at offset and length
   int numRead = zipInput.readFileDataPart(buffer);
   if (numRead < 0) { break; }
   // process bytes in the buffer
}
```

By default you will be reading the decoded (i.e. decompressed) bytes. You can also read the *raw* bytes, without conversion, using similar read methods with "raw" in the name.

```
// read _raw_ file data and write to file output-path
ByteArrayOutputStream baos = new ByteArrayOutputStream();
zipInput.readRawFileData(baos);
```

If you would like to stream the file-data out of the Zip file, you can open up an `InputStream` on the file-data either in encoded or raw mode. Calls to `read()` on the `InputStream` turn around and call the read methods on the `ZipFileInput`.

```
// reading from input stream calls thru to zipInput.readFileDataPart()
// or zipInput.readRawFileData() methods
InputStream inputStream =
    zipInput.openFileDataInputStream(false /* not raw */);
}
```

Opening an input-stream allows you to read a Zip file from within another Zip file – or a jar within a war, etc..

Once all of the data has been read for a particular file, there may be a `ZipDataDescriptor` entry written after the file data. This entry is read automatically by

the `ZipFileInput`. This descriptor is necessary in case the Zip file does not have the size or checksum/crc information at the start of the Zip file entry. See [File Buffering], page 6.

```
// return data-descriptor after file-data was read or null if none
ZipDataDescriptor dataDesc = zipInput.getCurrentDataDescriptor();
```

The descriptor holds the following information and is represented in SimpleZip by the class ZipDataDescriptor.java.

- crc32 checksum
- compressed size
- uncompressed size

Once all of the data has been read for a particular file and the optional descriptor has been read, you can then read the next header. See Section 2.2.2 [Read File Headers], page 3.

2.2.4 Reading Zip Central-Directory Entries

After all of the file headers and data in the Zip data, there are a series of central-directory entries written at the end of the Zip file which record extra information about each of the files and also provide the locations of the file-headers and data inside of the Zip file. You can read these entries if you would like.

```
// return next central-directory entry or null if none
ZipCentralDirectoryFileEntry directoryEntry =
    zipInput.readDirectoryFileEntry();
```

The central-directory file entries hold the following information for each file in the Zip. Some of the fields are duplicates of the fields in the file-header. The entries are represented by the class ZipCentralDirectoryFileEntry.java.

- format, version that wrote the entry
- format, version needed to decode
- general-purpose flags
- compression method
- last modified time
- last modified date
- crc32 checksum
- compressed size
- uncompressed size
- disk number start
- internal file attributes
- external file attributes
- relative offset of local header
- file-name bytes
- extra field metadata bytes
- comment bytes

If you have been reading file data directly out to disk using the `zipInput.readFileData(File)` method, you can modify the permissions on the file from the file-entry's using something like the following.

```
// read in a directory entry
directoryEntry = zipInput.readDirectoryFileEntry();
// assign file permissions according to previous entry
zipInput.assignDirectoryFileEntryPermissions(directoryEntry);
```

Once the `zipInput.readDirectoryFileHeader()` returns null then you are at the very end of the zip-file where there is some end information that can be read.

```
// read the end of entry of the zip-file
CentralDirectoryEnd directoryEnd = zipInput.readDirectoryEnd();
```

The end entry holds the following information.

- disk number
- disk number start
- num records on disk
- num records total
- directory size
- directory offset
- comment bytes

The SimpleZip class representing a central-directory end is ZipCentralDirectoryEnd.java.

2.3 Writing Zip Files

2.3.1 Constructing a ZipFileOutput

The main class that reads in Zip files is `ZipFileOutput`. You can write Zip data to a `File`, file-path string, or stream it out via an `OutputStream`.

```
// write to a file-path
ZipFileOutput zipOutput = new ZipFileOutput("/tmp/file.zip");
// write to a file
ZipFileOutput zipOutput =
    new ZipFileOutput(new File("/tmp/file.zip"));
// write to an OutputStream
ZipFileOutput zipOutput = new ZipFileOutput(outputStream);
```

The Zip file data starts with a file-header which contains (among other things) the compressed-size and checksum information that may not be known ahead of time. For files that are being deflated, these fields can be left as 0 in which case `ZipFileOutput` will write out a `ZipDataDescriptor` after the file data.

However, you can also turn on the buffering the file-data so we can calculate the compressed-size and crc checksum information *beforehand*, writing out a file-header with the size and checksum information filled in, removing the need for a `ZipDataDescriptor`.

```
// turn on buffering
zipOutput.enableFileBuffering(1024 * 1024 /* maxSizeBuffered */,
    100 * 1024 /* maxSizeInMemory */);
```

See the Javadocs for the enableFileBuffering(...) method for more information.

2.3.2 Writing File Header Entries

File headers immediately precede the file-data in a Zip. You need to first create a `ZipFileHeader` using the `ZipFileHeader.Builder` class.

```
// build our header by setting fields with with...() and set...()
ZipFileHeader fileHeader = ZipFileHeader.builder()
.withFileName("hello.txt")
.withGeneralPurposeFlags(GeneralPurposeFlag.DEFLATING_MAXIMUM)
.withLastModifiedDateTime(LocalDateTime.now())
.build();
// write the header to the zip output
zipOutput.writeFileHeader(fileHeader);
```

Even though the method is `writeFileHeader(...)`, the code may not write anything to disk immediately depending if buffering is enabled. Immediately after the header as been written, you should start writing the file-data.

2.3.3 Writing File Data to Buffer, File, or Stream

After writing the header you then write the file data. You can read in bytes to be written to the Zip file data from a file-path string, `File`, or stream it in via an `InputStream`.

```
// write bytes from file in specified path to the zip output
zipOutput.writeFileData("file.txt");
// write bytes from file to the zip output
zipOutput.writeFileData(new File("file.txt");
// stream bytes from an inputStream to the zip output
zipOutput.writeFileData(inputStream);
```

You can also have `ZipFileOutput` write file data from a series of buffers. You will need to call `finishFileData()` after all of the data is written.

```
// can also write at offset and length
zipOutput.writeFileDataPart(buffer);
zipOutput.writeFileDataPart(buffer);
// ... repeat until all bytes written
// after all bytes written you must call finish
zipOutput.finishFileData();
```

By default `ZipFileOutput` will take your bytes and write them to the Zip file encoded (i.e. deflate/comopress). You can also write the *raw* bytes without conversion using similar write methods with "raw" in the name.

```
// write _raw_ file data from the file specified by output-path
zipInput.writeRawFileData("file.txt");
...
```

If you would like to stream the file-data into the Zip file, you can open up an `OutputStream` for the file-data either in encoded or raw mode. Calls to `write()` on the `OutputStream` turn around and call the write methods on the `ZipFileOutput`.

```
// writing to output stream calls thru to zipOutput.writeFileDataPart()
// or zipOutput.writeRawFileData() methods
OutputStream outputStream =
    zipOutput.openFileDataOutputStream(false /* not raw */);
```

Opening an output-stream allows you to write a Zip file from within another Zip file – or a jar within a war, etc..

Once all of the data has been written for a particular file, the `ZipFileOutput` may automatically determine that it needs to write a `ZipDataDescriptor` entry with the sizes and crc checksum information.

2.3.4 Writing Central-Directory Entries

By default the `ZipFileOutput` will record the `ZipFileHeader` entries that have been written to the Zip output so they can be written out as the central-directory file-entries at the end of the Zip data. While you are writing each file, you have the option to associate more information with the file that will be written in each file-entry.

```
// add information to the file header that was just written that
// it is a text-file
zipOutput.addDirectoryFileInfo(
ZipCentralDirectoryFileInfo.builder().withTextFile(true).build());
```

There are a number of other fields that can be written. See the javadocs for the ZipCentralDirectoryFileInfo for more information.

At the very end of the Zip file the `ZipFileOutput` will automatically write the `ZipCentralDirectoryEnd` information. It will use fields from the `ZipCentralDirectoryFileInfo` as well to write out the fields.

2.4 Using With Maven

To use SimpleZip with maven, include the following dependency in your '`pom.xml`' file:

```
<dependency>
    <groupId>com.j256.simplezip</groupId>
    <artifactId>simplezip</artifactId>
    <version>2.1</version>
</dependency>
```

3 Various Parts of a Zip File

A Zip file is made up of the following pieces of information.

1. file information (0 or multiple)
   a. file header, see ZipFileHeader.java
      file-name
      flags
      compressed size
      uncompressed size
      checksum
      ...
   b. file data (encoded bytes)
   c. optional data-descriptor, either in standard or Zip64 format, see ZipDataDescriptor.java
      compressed size
      uncompressed size
      checksum
2. central-directory file entries (0 or multiple), see ZipCentralDirectoryFileEntry.java
   file-name
   file offset
   internal and external attributes
   file comment
   ...
3. optional Zip64 end, see Zip64CentralDirectoryEnd.java
   version made
   version needed
   ...
   extensible data
4. optional Zip64 end locator, see Zip64CentralDirectoryEndLocator.java
   disk number
   disk number start
   end offset
   number of disks
5. central-directory end (summary information), see ZipCentralDirectoryEnd.java
   number records
   central-directory offset
   comment
   ...

4 Example Code

Here is some example code to help you get going with SimpleZip. I often find that code is the best documentation of how to get something working. Please feel free to suggest additional example packages for inclusion here. Source code submissions are welcome as long as you don't get piqued if we don't chose your's.

- SimpleZipOutput.java

  This example writes out a Zip file with a file, a directory, and a file inside of that directory. See the source code.

- ZipFileCopy.java

  This example reads in a Zip file using ZipFileInput and writes it out with ZipFileOutput while hopefully not changing any of the internal structures. Please report a Zip that doesn't get copied perfectly with this. See the source code.

- ZipFileCopy.java

  This example reads in a Zip file and spits out the details about the file in excruciating detail. See the source code.

5 Open Source License

Index of Concepts