

Coffeearc Documentation

Version 0.75b (2010/06/12) © 2009-2010, Andreas Besting (andreas.besting@rwth-aachen.de)

Project site: <http://forge.simplana.de/projects/show/coffeearc>

General

About

Coffeearc is a simple, extensible GUI archiver that interfaces with file-to-file command-line compression tools or special plugins which use the Coffeearc API.

Bundled with CCM and Slug, both ultra-fast and strong compression (including high secure AES encryption with up to 256 bits) are its core features. Coffeearc supports some conventional archive formats like ZIP, TAR and others.

Coffeearc is free software released under the GNU General Public License (Version 3).

With Coffeearc you can:

- Archive your data using fast and strong compression and secure encryption.
- Transform virtually any command-line compression tool into a full-featured GUI archiver by writing a very simple XML-profile, and instantaneously benefit from
 - Easy GUI configuration
 - Solid-archiving
 - Multi-threading
 - Encryption
 - The possibility to combine *multiple* compression tools within a single profile and define which file types should be handled by which tool.
- Write a new Java plugin by extending two simple classes provided by the Coffeearc API to implement or test a new compression algorithm or to control the archive's data flow.

Some archiver features

The archiver's basic functionality is:

- Creating archives:
 - The archive's content are files and directories (including empty directories), which are either stored compressed, uncompressed or clustered in sub-archives. The archive's size is not limited and can exceed 4GB.
 - Coffeearc stores things like file attributes (read, write, exec) and other information (file size, date and – if specified – a checksum).
 - The compression speed can be increased by parallel execution of the compression tools on multi-core systems.
 - Archives can be encrypted with up to 256 bit (AES).
 - The compression rate can be improved by grouping together equal files to sub-archives

before compressing them. This is essentially solid archiving, but frequently applied to different subsets of files rather than all files at once. Therefore I will just call this *semi-solid* archiving (see below). You can define individual types of file-clusters in the XML-profile.

- Extracting archives:
 - Fully extract an archive
 - Extract only a subset of single files (or directories).
- Modifying archives, in detail:
 - Adding files to an archive.
 - Deleting entries from an archive (depending on the archive type).

The “.crc” / “crc.rae” archive format

The archives created by Coffeearc are actually uncompressed ZIP/ZIP64-files (supporting UTF-8) with the file extension “.crc” or “.crc.rae”. The latter file extension is used for encrypted archives. Information about the archive and the index of its entries are stored in an XML-file (archive.xml), which is compressed (using standard ZIP) and added as a sub-archive (index.crc_sub). Since all files are stored as default ZIP entries, even without the archive index each archive created with Coffeearc can always be extracted manually. So even without Coffeearc you will always be able to decompress your data using the following scheme:

- Extract the archive with your favorite archiver.
- Recursively loop through all extracted files:
 - Uncompress and rename each file that ends with “.crc_deflated” with the right command-line tool.
 - Rename all files that end with “.crc_inflated” (all these have been stored uncompressed).
- In case of a semi-solid archive:
 - Extract all sub-archives (which are now uncompressed) that match the file pattern “cluster.*.crc” (where * denotes any character sequence).

To restore things like file attributes or verify checksums, you'd have to use the information of the archive's index file “archive.xml”.

Semi-solid archives

As mentioned before, Coffeearc is able to cluster files which improves the compression rate in case of a large number of small files. It first creates uncompressed sub-archives before actually compressing them. With this I hoped to combine the advantages of solid and non-solid archives, since these “semi-solid” archives

- should provide similar compression rates in comparison to solid archives,
- can store uncompressed, compressed, single or clustered files,
- allow partial extraction with a less overhead
- are more secure in case they are damaged (you may loose a sub-archive, but not necessarily everything),
- allow multi-threading and

- can be handled more efficient when modifying contents.

User's guide

I believe, the user interface will mostly be self-explanatory. Therefore I will only describe the basic functionality here.

Starting Coffeearc

Since Coffeearc is written in Java, it should run on all platforms that have a Java Runtime 6 (or above). Under Windows, you can use the “coffeearc.exe”. Linux users may use “coffeearc.sh”. On all other platforms, a command like “java -jar coffeearc.jar” should do the trick.

Linux and WINE

If a Windows-profile (i.e. a profile that is created for a Windows exe file, like **Slug** and **CCM**) is used on Linux, Coffeearc tries to use WINE to execute the compression tools. If WINE can't be found, you have to specify the path to the WINE executable (e.g. on Ubuntu this should be /usr/bin).

Creating an archive

To create an archive you first have to create a list of files. To do that, just drag files or directories from the internal file browser (or a system file browser) into the table (note: Coffeearc will recursively scan all directories and add their content automatically to the file table). You can also use the “open” button for that. It's possible to extend the list or to remove files from the list at any time.

Note: Removing the files from the list will not delete the files from the filesystem. If you want to do the latter with Coffeearc, use the internal file browser and hit the “del” key.

After the file list is complete, select a profile, adjust the profile's options and hit the “create”-button. Then, follow the instructions. To create an encrypted archive, select the option “Encrypted archive” instead of “Standard archive”. Multi-threading options can be adjusted in case of a semi-solid archive and if supported by the profile.

Opening and extracting an archive

To view the archive's content, get the “crc”- or “crc.rae”-archive file into the table and double-click on it – Coffeearc then switches to the “archive-mode”. As a result, the archive's content is displayed and the “extract”-button is enabled. Before using that button, you may select a number of files to extract. You can also use the “search & select” function for that. If at least one file is selected, Coffeearc will prompt a dialog where you can choose the option to extract only the selected files. Otherwise, the archive will be fully extracted.

Note: Whenever accessing an encrypted archive, a dialog is prompted where you have to enter the right key.

Modifying an archive

When the archive is opened, you can add files to the archive by dragging them into the list. You can delete files from the archive by using the “remove”-button.

Note: Removing a directory entry is only possible if no file entries are stored under the directory.

Known issues

Here is a list of things Coffeecarc can't do at the moment or could have problems with:

- It's not checked whether an entry is already present in the list. So if you add the same file twice, it will be twice added to the archive.
- I noticed some performance issues when compressing a large number of small files using some fast command-line tools. Be sure to create a semi-solid archive in case this occurs.
- When switching the look&feel back to the Windows L&F, the GUI might miss some elements (just restart the program, this will fix it).

Keep in mind: Coffeecarc is currently still in development state!

Developer's guide

Requirements

There are three different ways of using Coffeecarc with external programs. The probably most common way is the following: Suppose you have a command-line compression tool “mytool.exe” that accepts some parameters like [options] [in] [out], where [in] is the input file, [out] is the output file and [options] denotes a set of options (e.g. “-c” for compression, “-d” for decompression). While the order and count of the parameters is not important, it is critical, that the compression tool produces exactly the file specified by [out], e.g. a command like

```
“mytool -c explorer.exe explorer.out”
```

should compress the file “explorer.exe” and produce the file “explorer.out”. If it appends something to the name or changes the file extension, it won't work. In case the compression succeeds, the tool should return with an exit value of 0 (the same applies for decompression, of course).

Another way of getting Coffeecarc to work are pipes. This is actually a better method than the first one, but can only be applied only if supported by the compression tool. I will explain this later.

Last but not least Coffeecarc has an easy Java class interface that can be extended. If you want something special, this gives you complete control over the data flow into and out of an archive by using buffered streams. This will also be explained later.

Creating a profile

To get a tool to work with Coffeecarc, you have to create an XML-profile and then copy the tool and the XML-file in the “profiles” directory, which is located in the program directory. Following the last example, a simple working profile for “mytool.exe” would look like this:

```
<profile name="mytool" version="1.0">
  <gui name="Mode" type="check">
    <option label="Fast" selected="yes">
      <replace key="%opt1" with="-f"/>
    </option>
  </gui>
  <encode file="mytool.exe" params="-c %in %out %opt1">
    <decode params="-d %in %out"></decode>
  </encode>
</profile>
```

The profile's name and version are stored inside each archive. So, if you want to extract (or update) an archive, you need a fully compatible profile that matches these parameters. The “encode” and

“decode” tags define the parameters needed for compression and decompression, respectively. The strings “%in” and “%out” will be automatically replaced by Coffeearc by the right filenames. If one or more of these parameters are missing, Coffeearc will work in “pipe-mode”, which is another nice way of linking the archiver to a compression tool (see below).

Along with selecting this example profile, Coffeearc creates a small panel with a single (preselected) checkbox. When the checkbox is selected, the parameter “-f” will be passed to the compression tool (by replacing the “%opt1% placeholder).

Saving this in a file “profiles\mytool.xml” (the name is not important, but the extension must be “xml”) will get the archiver to work fine, but you might want to add additional tags to enable some nice features of Coffeearc. This will be explained next.

Important profile features

You can specify the following tags:

- <description>: A description of the profile.
- <gui>: GUI Elements like check-boxes, radio-buttons or combo-boxes.
- <skip>: A list of file extensions, these files are stored uncompressed.
- <checksum>: A flag that enables Coffeearc to calculate checksums for each entry.
- <multithreading>: Allows the use of multithreading (compression only). See further information below.
- <solid>: A flag that allows the creation of semi-solid archives. Also, the definition of individual sets of file extensions to bundle together, is possible.
- Additional <encode> tags can be specified and linked with clusters to combine several compression tools in one profile.
- <sort>: Sort files (by name, extension, size). This tag is for example used in the .tar profiles to optimize the compression rate.

To see how to use these features, just take a look at the profile example file “profiles\profile.xml”. This example contains additional comments and covers all the listed features above, in detail.

Using Coffeearc in pipe-mode

Enabling pipes in a profile is an alternative way of interfacing Coffeearc with a compression tool. If pipes are supported by the tool, their usage will result in the use of lesser temporary files (or even no temporary files at all, in case of a non-semi-solid archive), which should speed up the compression and decompression, respectively.

Configuring the profile for pipes is no more difficult than for the default mode: If one or both of the placeholders for the input- or output files inside the “params” attribute of the encode (or decode) tag are missing, Coffeearc expects the compression tool to

- read data from its process' input stream if the “%in” placeholder is missing (Coffeearc will then write data to the piped process' output stream)
- write data to its process' output stream if the “%out” placeholder is missing (Coffeearc will then read data from the piped process' input stream)

For example, suppose “mytool.exe” could be configured to write compressed data to its process' output stream (instead of a file) by replacing “%out” (i.e. the output filename) by a “:”. Further, reading uncompressed data from the input stream while decompressing could be done by replacing “%in” (i.e. the input filename) by a “:”. Then the “encode” tag of the example profile above would

look like this:

```
<encode file="mytool.exe" params="-c %in : %opt1">
  <decode params="-d : %out"></decode>
</encode>
```

In this case, no temporary files are needed and Coffeearc can write directly into the archive while compressing and while decompressing, data is also directly passed to the decompression routine. Since Coffeearc can simultaneously read from and write to a process' in- and output stream, both cases are possible as well:

```
<encode file="mytool.exe" params="-c : : %opt1">
  <decode params="-d : : "></decode>
</encode>
```

For the sake of completeness, even the cases “: %out” (encode) and “%in :” (decode) are implemented, but in each case temporary files will still be produced.

Enabling multi-threading

Since compression can take time and probably most single-file compression tools don't implement parallelization, the archiver itself is a good place to do just that. The idea is quite simple, Coffeearc just tries to launch more than one compression thread/process at once. However, by default this feature is **disabled** because not in every case it makes sense to use it and not properly configured, it might do more harm than good.

For example, this feature is certainly not interesting for ultra-fast compression tools (like Slug) since this could really kill the HDD performance. However, in cases where the CPU load is high and HDD load is low, this will push the compression speed (parallel decompression isn't supported yet), provided the system has more than one CPU core, of course. Coffeearc does not synchronize the start of the compression threads which means that the subroutines are called whenever the next cluster is sorted and ready to be compressed. As a result, it could take some time at the beginning for the parallelization to work. If the compression is faster than the gathering of files, there might also be gaps where the CPU load drops below 100%. To optimize this process, you should choose a not too small minimum and not too large maximum bound for a cluster size. I haven't tested it much out, but a setting like

```
<solid min="5" max="50"/>
```

(or 10/100) seems reasonable. I used this setting for CCM.

Switching on multi-threading can simply be done by inserting the `<multithreading/>` tag into the profile. The `maxlevel` attribute allows you to control the maximum number of threads selectable by the user when creating an archive. The default value is 0 (all levels allowed), 1 means the maximum level is background-sorting, 2 and above point out the number of maximum selectable parallel compression threads. See the example profile 'profile.xml' for more information.

Some notes: Parallelization only works for semi-solid archives! Using this feature usually results in more temporary cluster files. I plan to improve this drawback one day for the pipe-mode, but this requires some major refactoring.

Text-output

While operating with command-line tools, Coffeearc redirects the output of the tool(s) to the console. Both the process' input and -error-stream are read out and passed to the console (in pipe-mode, only the error-stream is used)

Preconfigured command-line profiles

Coffeecarc comes with two compression tools a friend of mine wrote: **Slug** and **CCM**, two command-line tools designed for ultra-fast and strong compression. These are my favorite tools and part of the reason why I wrote Coffeecarc.

Writing a new Java-plugin – how and why

A Java-based plugin can be created by using the Coffeecarc API. This doesn't mean that a compression algorithm has to be implemented in Java. You could for example use the interface to just implement a data filter or to redirect data to a native compression tool in a way that is not supported by Coffeecarc by default (e.g. you could interface with a native library by using the Java native interface).

So, by writing a Java plugin you gain more control of the data flow and properly done, your profile could work on all platforms with a Java VM. Besides, a Java plugin uses direct input- and output-streams and therefore, Coffeecarc creates less temporary files. Each call to the “encode” and “decode” method contains information of the file- or cluster-type, so that you always know what data you're about to process.

Here is a quick recipe on how to create a Java-plugin:

1. Download Eclipse (www.eclipse.org).
2. Create a new Java project and add the files “coffeecarc.jar” and “libs/truezip-6.jar” to the build path (project properties).
3. Create the encoder class: Extend the class “StreamEncoder” and implement the “compress” method (look at the Coffeecarc source if you need more information).
4. For the decoder, do the same with “StreamDecoder” (and the decompress method).
5. Export your project as a JAR file (important: use the manifest file “plugin.mf”).

As a last step, you have to create an XML-profile similar to the one described for command-line tools. The only difference is that you have to add attributes that point to the right classes. Here is an example:

```
<profile name = "Store" version="1.0">
  <encode file="store.jar" class="store.Encoder">
    <decode class="store.Decoder"/>
  </encode>
</profile>
```

This profile is the actual “Store” profile of Coffeecarc. Note that the classes “Encoder” and “Decoder” are both stored in the package “store”. For further details, take a look at source code and profile files of the “Store” and “CoffeeZip” plugins.

Troubleshooting

Here are a few things you can do, if something does not work as expected:

- When creating a new profile, keep attention to errors displayed in the Coffeecarc console right after you launch the program.
- Keep attention to any exceptions thrown (these will also be redirected to the console).
- You can write to me if you think you found a bug (or any other strange behavior). Please read the “known issues” below, before you do that.

Future development

In future versions the highest priority will be to maintain compatibility with all archives created with older versions.