I'm not a robot

Unrar is fully compatible with multiple compression archive formats, whether they use an open source or proprietary compression algorithm. Compatible formats include: ARJ Compressed File Archive Microsoft Cabinet Archive File Compiled HTML File Archive Unix General File Archive Hierarchical File System Archive Microsoft Windows Installer File Nullsoft Scriptable Install System WinRAR Compressed Archive Red Hat Package Manager File Unix/Linux Tape Archive File Universal Disk Format File Windows Imaging Format File Extensible Archive Format File Unrar is a file archiver – just like WinZip or WinRAR. Unrar can open compressed files or compress files. The best part: Unrar is completely FREE. Have you ever received a file attachment that you couldn’t open? Or maybe you need to compress a file to save space on your hard drive or so you can attach it to an email? Unrar is the perfect solution! With Unrar, you can open archived files or compress files into one or groups of files. Everyone should have this free, easy to use file archiving solution! Experience Unrar – Download Now! Unrar is compatible with all file formats! Open or compress whatever files you need. Unrar is faster & 70% more efficient than WinZip & WinRAR. Unrar is FREE! Save up to $30 by using Unrar instead of WinZIP or WinRAR. Project description Project details Release history Download files Work with RAR archive files through unrar library using ctypes. Install UnRAR library python-unrar requires UnRAR library. You can download UnRAR library sources from: and compile (you may need to rename the makefile that you want to use according to your OS) and install it from there: $ make lib $ make install-lib (latest tested sources: For Windows you can also download the already compiled library ( . If you prefer not to install the library, you should make it "findable" by adding the library file to a directory where libraries are searched (or change required environment variable). As an alternative, you can also set UNRAR_LIB_PATH variable in your environment to the library path and python-unrar will try to load the UnRAR library from there. Testing without installing You can build from source with: $ python setup.py build And then run a Python shell with unrar available: $ PYTHONPATH=`pwd`/build/lib python Or you could also directly add the unrar directory from this repo to your PYTHONPATH. In any case you will still need to make the unrar library available as mentioned above. Install python-unrar To install python-unrar: $ pip install unrar Examples >>> from unrar import rarfile >>> rar = rarfile.RarFile('sample.rar') >>> rar.namelist() [u'test_file.txt'] >>> rar.printdir() File Name Modified Size test_file.txt 2013-04-14 08:20:28 17 >>> rar.testrar() >>> info = rar.infolist()[0] >>> info.filename u'test_file.txt' >>> info.file_size 17L >>> info.date_time (2013L, 4L, 14L, 8L, 20L, 28L) >>> rar.extractall() >>> rar.read('test_file.txt') 'This is for test.' Docs Check full documentation in . Project details Download the file for your platform. If you're not sure which to choose, learn more about installing packages. Source Distribution unrar-0.4.tar.gz (36.7 kB view details) Built Distribution unrar-0.4-py3-none-any.whl (25.3 kB view details) Details for the file unrar-0.4.tar.gz. File metadata Download URL: unrar-0.4.tar.gz Upload date: Sep 8, 2019 Size: 36.7 kB Tags: Source Uploaded using Trusted Publishing? No Uploaded via: twine/1.14.0 pkginfo/1.5.0.1 requests/2.22.0 setuptools/41.2.0 requests-toolbelt/0.9.1 tqdm/4.35.0 CPython/3.7.3 Hashes for unrar-0.4.tar.gz Algorithm Hash digest SHA256 b24447a5b93024be600ef8255668ba23a30f451176577b691559ea1359f7d164 MD5 6d78245a417fb97e48029fefc8b41eec BLAKE2b-256 ead6b53a4216d73a10602f524cf728c3c272b2e1f524b62f46589303225564c0 See more details on using hashes here. Details for the file unrar-0.4-py3-none-any.whl. File metadata Download URL: unrar-0.4-py3-none-any.whl Upload date: Sep 8, 2019 Size: 25.3 kB Tags: Python 3 Uploaded using Trusted Publishing? No Uploaded via: twine/1.14.0 pkginfo/1.5.0.1 requests/2.22.0 setuptools/41.2.0 requests-toolbelt/0.9.1 tqdm/4.35.0 CPython/3.7.3 Hashes for unrar-0.4-py3-none-any.whl Algorithm Hash digest SHA256 0ce03e28bfa90415ee1a92c8d38cad8ea244427a59a8a39c0d42bac8727cfaa0 MD5 1a7301a7f9238a200941f8139fd18be5 BLAKE2b-256 bb0b53130ccd483e3db8c8a460cb579bdb21b458d5494d67a261e1a5b273fbb9 See more details on using hashes here. High-level wrapper around the unrar C library provided by rarlab. This library can only extract and list archives, it cannot create them. Please look inside the examples directory to see how to use this library. Specifically the lister example is well documented and advanced! Basic example to list archive entries: use Archive; Run this example: cargo run --example basic_list path/to/archive.rar. You can create an archive by using the rar CLI: rar a archive.rar . Overview The primary type in this crate is [Archive] which denotes an archive on the file system. Archive itself makes no FS operations, unless one of the open methods are called, returning an [OpenArchive]. Archive The [Archive] struct provides two major classes of methods: methods that do not touch the FS. These are opinionated utility methods that are based on RAR path conventions out in the wild. Most commonly, multipart files usually have extensions such as .part08.rar or .r08.rar. Since extracting must start at the first part, it may be helpful to figure that out using, for instance, archive.as_first_part() methods that open the underlying path in the specified mode (possible modes are [List], [ListSplit] and [Process]). These methods have the word open in them, are fallible operations, return [OpenArchive] inside a Result and are as follows: OpenArchive An archive is opened in one of these three modes: [List], [ListSplit] or [Process]. This library does not provide random access into archives. Instead, files inside the archive can only be processed as a stream, unidirectionally, front to back, alternating between ReadHeader and ProcessFile operations (as dictated by the underlying C++ library). That is the idea behind cursors: OpenArchive: Cursors Via cursors, the archive keeps track what operation is permitted next: [CursorBeforeHeader] -> ReadHeader [CursorBeforeFile] -> ProcessFile The library enforces this by making use of the typestate pattern. An archive, once opened, starts in the CursorBeforeHeader state and, thus, must have its read_header method called, which returns a new OpenArchive instance in the CursorBeforeFile state that only exposes methods that internally map to the ProcessFile operation. Which methods are accessible in each step depends on the archive's current state and the mode it was opened in. Available methods for Open mode/Cursor position combinations Here is an overview of what methods are exposed for the OpenMode/Cursor combinations: OpenArchive: Iterator Archives opened in [List] or [ListSplit] mode also implement [Iterator] whereas archives in [Process] mode do not (though this may change in future releases). That is because the first two will read and return headers while being forced to skip over the payload whereas the latter has more sophisticated processing possibilities that's not easy to convey using an [Iterator]. Example For more sophisticated examples, please look inside the examples/ folder. Here's what a function that returns the first content of a file could look like: # use Path; # use ; # # let data = first_file_content.unwrap; # assert_eq!; Features As this library is only a wrapper, these following features are not easily feasible and as such not planned: Creating archives Random access into arbitrary archive entries Pure Rust implementation Processing archives from a file descriptor / fs::File handle Processing archives from a byte stream Contributing Feel free to contribute! If you detect a bug, open an issue. Pull requests are also welcome! Help If you need help using the library, feel free to create a new discussion or open an issue. License The parts authored by this library's contributors are licensed under either of at your option. The embedded C/C++ library uses its own license. For more informations, see its license file. You can’t perform that action at this time. Page 2 You can’t perform that action at this time. Page 3 You can’t perform that action at this time. Page 4 You can’t perform that action at this time. Expand description High-level wrapper around the unrar C library provided by rarlab. This library can only extract and list archives, it cannot create them. Please look inside the examples directory to see how to use this library. Specifically the lister example is well documented and advanced! Basic example to list archive entries: use unrar::Archive; fn main() { for entry in Archive::new("archive.rar").open_for_listing().unwrap() { println!("{}", entry.unwrap()); } } Run this example: cargo run --example basic_list path/to/archive.rar. You can create an archive by using the rar CLI: rar a archive.rar . §Overview The primary type in this crate is Archive which denotes an archive on the file system. Archive itself makes no FS operations, unless one of the open methods are called, returning an OpenArchive. §Archive The Archive struct provides two major classes of methods: methods that do not touch the FS. These are opinionated utility methods that are based on RAR path conventions out in the wild. Most commonly, multipart files usually have extensions such as .part08.rar or .r08.rar. Since extracting must start at the first part, it may be helpful to figure that out using, for instance, archive.as_first_part() methods that open the underlying path in the specified mode (possible modes are List, ListSplit and Process). These methods have the word open in them, are fallible operations, return OpenArchive inside a Result and are as follows: §OpenArchive An archive is opened in one of these three modes: List, ListSplit or Process. This library does not provide random access into archives. Instead, files inside the archive can only be processed as a stream, unidirectionally, front to back, alternating between ReadHeader and ProcessFile operations (as dictated by the underlying C++ library). That is the idea behind cursors: §OpenArchive: Cursors Via cursors, the archive keeps track what operation is permitted next: CursorBeforeHeader -> ReadHeader CursorBeforeFile -> ProcessFile The library enforces this by making use of the typestate pattern. An archive, once opened, starts in the CursorBeforeHeader state and, thus, must have its read_header method called, which returns a new OpenArchive instance in the CursorBeforeFile state that only exposes methods that internally map to the ProcessFile operation. Which methods are accessible in each step depends on the archive’s current state and the mode it was opened in. §Available methods for Open mode/Cursor position combinations Here is an overview of what methods are exposed for the OpenMode/Cursor combinations: §OpenArchive: Iterator Archives opened in List or ListSplit mode also implement Iterator whereas archives in Process mode do not (though this may change in future releases). That is because the first two will read and return headers while being forced to skip over the payload whereas the latter has more sophisticated processing possibilities that’s not easy to convey using an Iterator. §Example For more sophisticated examples, please look inside the examples/ folder. Here’s what a function that returns the first content of a file could look like: fn first_file_content(path: P) -> UnrarResult { let archive = Archive::new(&path).open_for_processing()?; let archive = archive.read_header()?.expect("empty archive"); dbg!(&archive.entry().filename); let (data, _rest) = archive.read()?; Ok(data) } §Features §Non-Features As this library is only a wrapper, these following features are not easily feasible and as such not planned: Creating archives Random access into arbitrary archive entries Pure Rust implementation Processing archives from a file descriptor / fs::File handle Processing archives from a byte stream §Contributing Feel free to contribute! If you detect a bug, open an issue. Pull requests are also welcome! §Help If you need help using the library, feel free to create a new discussion or open an issue. §License The parts authored by this library’s contributors are licensed under either of Apache License, Version 2.0 (LICENSE-APACHE or MIT license (LICENSE-MIT or at your option. The embedded C/C++ library uses its own license. For more informations, see its license file. pub use error::UnrarResult;errorArchiveA RAR archive on the file system.CursorBeforeFileType parameter for OpenArchive denoting a process_file operation must follow next.CursorBeforeHeaderType parameter for OpenArchive denoting a read_header operation must follow next.FileHeaderMetadata for an entry in a RAR archiveListAn OpenMode for listing RAR archive entries.ListSplitAn OpenMode for listing RAR archive entries.OpenArchiveAn open RAR archive that can be read or processed.ProcessAn OpenMode for processing RAR archive entries.VolumeInfoVolume information on the file that was initially opened. The ‘unrar’ command is a powerful utility designed to handle RAR archive files, which are widely used for compressing and packaging data into a single file, making it easier to store or distribute. The command is versatile, providing various options to extract, list, and test archive contents. It offers a straightforward approach to managing RAR files, whether conserving the original directory structure or extracting contents directly to a new location.Code:Motivation: When dealing with RAR archives, it’s crucial to preserve the original directory structure. This use case is ideal for users who want to maintain the same hierarchy within the extracted files as was present during the archiving process. It ensures that any folders and subdirectories are recreated exactly as they were, which is often necessary for applications where directory paths and file locations are significant.Explanation:unrar: Invokes the unrar command-line utility.x: This switch tells unrar to extract files, fully preserving the paths present in the RAR archive.compressed.rar: This is the archive file from which contents need to be extracted. The file may be either relative to your current directory or provide a full path.Example Output:Extracting path/to/file1.txt OK Extracting path/to/file2.txt OK Creating path/to/subdir/ Extracting path/to/subdir/file3.txt OK Code:unrar x compressed.rar path/to/extract Motivation: There are instances where you want to extract archive contents to a specific directory rather than the current working directory. Such a scenario could arise when organizing files from multiple archives into a single directory, or when you need to prepare files for a different environment or system without altering the existing file structure.Explanation:unrar: The command to run the tool.x: Option to extract files with full path creation.compressed.rar: Specifies the path to the RAR archive.path/to/extract: Designates the directory where you want the files to be extracted. If it doesn’t exist, unrar may attempt to create it, depending on permissions.Example Output:Extracting file1.txt OK Extracting file2.txt OK Creating path/to/extract/path/to/subdir/ Extracting path/to/subdir/file3.txt OK Code:Motivation: Sometimes, preserving the directory structure is unnecessary, and extracting files directly into the current directory simplifies file access. This use case applies when files within the archive can operate independently of their original path contexts, or when a quick access without directory management is required.Explanation:unrar: Command to execute the tool.e: Command line switch to extract files, flattening the directory structure.compressed.rar: The archive file whose contents are to be extracted.Example Output:Extracting file1.txt OK Extracting file2.txt OK Extracting file3.txt OK Code:Motivation: Before extracting files, particularly from unknown or secure sources, verifying the integrity of the archive ensures that all contents are uncorrupted and safe to use. This use case is highly applicable in scenarios where data integrity is critical, such as software distributions or sensitive data storage.Explanation:unrar: Command line to launch the extraction tool.t: Option to test the archive for any file damages or corruptions.compressed.rar: RAR archive to be tested.Example Output:Testing file1.txt OK Testing file2.txt OK Testing file3.txt OK All OK Code:Motivation: Listing the contents before extracting files can be useful for assessing the contents of the archive, estimating file sizes, or checking for specific files. This command is beneficial when you need to make decisions about which files to extract or handle, without compromising time or resources to extract unnecessary data.Explanation:unrar: The command initiating the operation.l: Switch to list files and their details within the archive.compressed.rar: Specifies the RAR archive to be examined.Example Output:Archive: compressed.rar Details Path/To/FileName Size Date Time ------- ---------- ---- ---- ---- file1.txt 2048 2023-09-01 10:00 file2.txt 1024 2023-09-02 11:00 path/to/file3.txt 4096 2023-09-03 12:00 Conclusion:The ‘unrar’ command is an essential tool for working with RAR archives, offering a wide range of functionality that makes file management versatile and accessible. Whether preserving directory structures, ensuring data integrity, or simply inspecting archive contents, ‘unrar’ handles these tasks efficiently and effectively. Understanding and utilizing these use cases allows users to streamline file extraction and management processes. You can’t perform that action at this time.