fast-package-file

Release 1.1

Jul 14, 2019

Contents

Package a directory to a file, with fast file access and compression support

```
import fast_package_file

# Package a directory into a file
fast_package_file.build('a_directory', 'a_package.file')

# Prepare a package file
data_package = fast_package_file.PackagedDataFile('a_package.file')

# Load a file from the packed directory and save it
with open('any.file', 'wb') as any_file:
        data_package.load_file('path\\to\\any.file')

# Or just get the raw binary data
from PIL import Image
i = Image.open(io.BytesIO(data_package.load_file('image.png')))

# Some other useful functions
data_package.load_bulk(prefix='audio\\sfx\\', postfix='.wav')
fast_package_file.oneshot('a_package.file', 'path\\to\\any.file')
fast_package_file.oneshot_bulk('a_package.file', prefix='audio\\sfx\\', postfix='.wav
↪')
```

CHAPTER 1

Installation

From PyPI:

```
pip install fast-package-file
```

Or from Github:

```
pip install git+git://github.com/Kataiser/fast-package-file.git@master#egg=fast_
↪package_file
```

CHAPTER 2

Features

- Is fast because only the data needed is loaded from the package file, total package size is irrelevant
- Obfuscates files from (most) users
- Like a .zip file, but doesn't decompress the entire thing when reading just one file
- Includes the entire directory and subdirectories, not just surface-level files
- Files are compressed with Gzip, but only if compression improves file size (per file) and is enabled (per package file)
- Pretty good error handling when loading package files, just catch fast_package_file.PackageDataError
- A simple, open-source and documented file format that can easily be parsed and read in other languages
- Inspired by video game packaging, such as UE4's .pak or GTA V's .rpf formats
- Cross-platform, has CI for Linux, MacOS, and Windows

CHAPTER 3

Usage guide

3.1 Function reference

3.1.1 Building packages

`fast_package_file.`**`build`**(*directory: str, target: str, compress: bool = True, keep_comp_threshold: float = 0.98, hash_mode: Optional[str] = None, comp_func: Callable[[bytes], bytes] = None, crc32_paths: bool = False, progress_bar: bool = True, silent: bool = False*)

Build a packaged data file from a directory.

Parameters

- **`directory`** – The directory to package. Includes all subdirectories.
- **`target`** – The path for the package file. If it already exists, it's overwritten. The file extension can be whatever you like.
- **`compress`** – Whether to compress the package, either with `comp_func` or Gzip by default.
- **`keep_comp_threshold`** – 0 through 1 (default is 0.98). For each input file, if compression doesn't improve file size by this ratio, the file is instead stored uncompressed. Set to 1 to compress every file no matter what.
- **`hash_mode`** – The hash method to use to ensure file validity. Can be "md5", "crc32", or "sha256". If `None` (the default), only the first and last bytes are compared.
- **`comp_func`** – A supplied decompression function that takes `bytes` and returns `bytes`. Some recommendations: LZMA, LZMA2, Deflate, BZip2, Oodle, or Zstandard.
- **`crc32_paths`** – Store file paths as crc32 numbers. Useful for obfuscating file names and paths.
- **`progress_bar`** – Whether to show a progress bar (uses tqdm). If tqdm isn't installed, this is irrelevant.

- **silent** – Disable all prints.

3.1.2 Getting data out of packages

fast_package_file.PackagedDataFile.**__init__**(*self, data_file_path: str, prepare: bool = True, decomp_func: Callable[[bytes], bytes] = None*)

Prepare a packaged data file.

Parameters

- **data_file_path** – Location of the package.
- **prepare** – Whether to load, decompress (if necessary), and parse the file location header now, or wait until the first file is loaded. Regardless, the entire package is not loaded.
- **decomp_func** – A supplied decompression function.

fast_package_file.PackagedDataFile.**load_file**(*self, file: str*) → bytes

Load a single file from the package. Also loads and parses the location data header for the packge, if it hasn't been already.

Parameters **file** – The path to the file, relative to the input directory (e.g. a file at surface level would be `file.txt`, and one folder in would be `folder\file.txt`).

Returns The file as a bytes object, uncompressed.

Note: File paths stored within a package file are modified to always use backslashes as path seperators, regardless of what OS is used to build or load the package. Be sure to either escape the backslashes or use raw strings.

---

fast_package_file.PackagedDataFile.**load_bulk**(*self, prefix: str = '', postfix: str = ''*) → Dict[str, bytes]

Load multiple files at once, based on a prefix and/or a postfix for the file path (uses `.startswith` and `.endswith`).

Parameters

- **prefix** – File path prefix (e.g. a subdirectory).
- **postfix** – File path postfix (e.g. a file extension).

Returns A `dict`, formatted as `{'path': bytes}`.

Note: Doesn't support packages using crc32 file paths.

3.1.3 Helpers

fast_package_file.**oneshot**(*data_file_path: str, file: str, decomp_func: Callable[[bytes], bytes] = None*) → bytes

Load a single file from a package file.

Parameters

- **data_file_path** – Location of the package.
- **file** – The path to the file, relative to the input directory (same as *load_file()*).

- **decomp_func** – A supplied decompression function.

Returns The file as `bytes`, uncompressed.

Note: If you're planning on ever loading another file from the same package, it's recommended to use `PackagedDataFile` explicitly since it caches the file location data.

`fast_package_file.`**`oneshot_bulk`**(*data_file_path: str, prefix: str = '', postfix: str = '', decomp_func: Callable[[bytes], bytes] = None*) → Dict[str, bytes]

Combines *`oneshot()`* and *`load_bulk()`*. Same note as *`oneshot()`*.

Parameters

- **data_file_path** – Location of the package.
- **prefix** – Same as *`load_bulk()`*.
- **postfix** – Same as *`load_bulk()`*.
- **decomp_func** – A supplied decompression function.

Returns A `dict`, formatted as `{'path': bytes}`.

Note: Doesn't support packages using crc32 file paths.

`fast_package_file.PackagedDataFile.`**`file_data`**

A dictionary containing the file location data.

Type `dict: {'path': (offset, length, compressed (1 or 0), first byte, last byte)}`

`fast_package_file.PackagedDataFile.`**`__repr__`**(*self*)

Includes path, number of files, and total file size.

Returns `str`

exception `fast_package_file.`**`PackageDataError`**

3.2 Packaged data file format

Although the builder and loader for this format are implemented in Python, the format can of course be read by any languange.

- `0x00` (2-byte unsigned little-endian int): File format version.
- `0x01` (8-byte unsigned little-endian int): Size of the file location header, in bytes, as stored in the file (i.e. after compression).
- `0x09` (1-byte bool): Whether the header is compressed (not including these first 10 bytes, which are never compressed).
- `0x0A` (1-byte bool): Whether the file paths use crc32 encoding.
- `0x0B` (UTF-8 string): The file location header, as JSON.
- The rest is file data, placed end-to-end.

3.2.1 File location header (JSON)

```
{"folder\\file1.txt":
        [file location, relative to the end of the entire header (int),
         file size, after compression if enabled (int),
         file is compressed (bool),
         first byte of file (int),
         last byte of file (int)],
"folder\\file2.txt": [...], ...}
```

Note: This example is multi-line for readability, but the actual format has no newlines.

Note: File paths are stored as actual double backslashes (\\). Python's JSON loader handles this automatically, make sure yours does or reformat the string.

Note: If using crc32 file paths, they are stored as strings of integers.

CHAPTER 4

Contribute

- Issue Tracker: https://github.com/Kataiser/fast-package-file/issues
- Source Code: https://github.com/Kataiser/fast-package-file

CHAPTER 5

License

The project is licensed under the MIT license.

Python Module Index

f

Index

Symbols

B

F

L

O

P