BitFileCompressor

Jump to bottom

Oz edited this page on Sep 26, 2024 · 8 revisions

The BitFileCompressor class allows compressing files and directories. The compressed archives can be saved to the filesystem, standard streams, or memory buffers.

```
#include <bit7z/bitfilecompressor.hpp>
```

inherits from ***BitCompressor****< const* ***tstring*** *& >.*

List of all members

Public Members

Return type	Name
	BitFileCompressor(const Bit7zLibrary& lib, const BitInOutFormat& format)
void	clearPassword() noexcept
void	compress(const std::map< tstring, tstring >& inPaths, const tstring& outFile) const
void	compress(const std::map< tstring, tstring >& inPaths, std::ostream& outStream) const
void	compress(const std::vector< tstring >& inPaths, const tstring& outFile) const
void	compress(const std::vector< tstring >& inPaths, std::ostream& outStream) const
void	compressDirectory(const tstring& inDir, const tstring& outFile) const
void	compressDirectoryContents(const tstring& inDir, const tstring& outFile, bool recursive = true, const tstring& filter

Return type	Name
	= "*") const
void	compressFile(const tstring& inFile, const tstring& outFile, const tstring& inputName = {}) const
void	compressFile(const tstring& inFile, ostream& outStream, const tstring& inputName = {}) const
void	compressFile(const tstring& inFile, vector< byte_t >& outBuffer, const tstring& inputName = {}) const
void	compressFiles(const std::vector< tstring >& inFiles, const tstring& outFile) const
void	compressFiles(const tstring& inDir, const tstring& outFile, bool recursive = true, const tstring& filter = "*") const
const BitInOutFormat &	compressionFormat() const noexcept
BitCompressionLevel	compressionLevel() const noexcept
BitCompressionMethod	compressionMethod() const noexcept
bool	cryptHeaders() const noexcept
uint32_t	dictionarySize() const noexcept
FileCallback	fileCallback() const
const BitInFormat &override	format() const noexcept
bool	isPasswordDefined() const noexcept
const Bit7zLibrary &	library() const noexcept
OverwriteMode	overwriteMode() const
tstring	password() const
PasswordCallback	passwordCallback() const
ProgressCallback	progressCallback() const
RatioCallback	ratioCallback() const
bool	retainDirectories() const noexcept
void	setCompressionLevel(BitCompressionLevel level) noexcept
void	setCompressionMethod(BitCompressionMethod method)

Return type	Name
void	setDictionarySize(uint32_t dictionarySize)
void	setFileCallback(const FileCallback& callback)
void	setFormatProperty(const wchar_t(&name)[N], const T& value) noexcept
void	setFormatProperty(const wchar_t(&name)[N], T value) noexcept
void	setOverwriteMode(OverwriteMode mode)
void	setPassword(const tstring& password) override
void	setPassword(const tstring& password, bool cryptHeaders)
void	setPasswordCallback(const PasswordCallback& callback)
void	setProgressCallback(const ProgressCallback& callback)
void	setRatioCallback(const RatioCallback& callback)
void	setRetainDirectories(bool retain) noexcept
void	setSolidMode(bool solidMode) noexcept
void	setStoreSymbolicLinks(bool storeSymlinks) noexcept
void	setThreadsCount(uint32_t threadsCount) noexcept
void	setTotalCallback(const TotalCallback& callback)
void	setUpdateMode(bool canUpdate)
void	setUpdateMode(UpdateMode mode)
void	setVolumeSize(uint64_t volumeSize) noexcept
void	setWordSize(uint32_t wordSize)
bool	solidMode() const noexcept
bool	storeSymbolicLinks() const noexcept
uint32_t	threadsCount() const noexcept
TotalCallback	totalCallback() const
UpdateMode	updateMode() const noexcept
uint64_t	volumeSize() const noexcept
uint32_t	wordSize() const noexcept

Member Function Documentation

BitFileCompressor(const Bit7zLibrary& lib, const BitInOutFormat& format)

Constructs a BitFileCompressor object. The Bit7zLibrary parameter is needed to have access to the functionalities of the 7z DLLs. On the contrary, the BitInOutFormat is required to know the format of the output archive.

Parameters:

- **lib**: the 7z library used.
- **format**: the output archive format.

void clearPassword() noexcept

Clear the current password used by the handler. Calling clearPassword() will disable the encryption/decryption of archives.

> **Note**
>
> This is equivalent to calling setPassword(L"").

void compress(const std::map< tstring, tstring >& inPaths, const tstring& outFile) const

Compresses the given files or directories using the specified aliases. The items in the first argument must be the relative or absolute paths to files or directories existing on the filesystem. Each pair in the map must follow the following format: {"path to file in the filesystem", "alias path in the archive"}.

Parameters:

- **inPaths**: a map of paths and corresponding aliases.
- **outFile**: the path (relative or absolute) to the output archive file.

void compress(const std::map< tstring, tstring >& inPaths, std::ostream& outStream) const

Compresses the given files or directories using the specified aliases. The items in the first argument must be the relative or absolute paths to files or directories existing on the filesystem. Each pair in the map must follow the following format: {"path to file in the filesystem", "alias path in the archive"}.

Parameters:

- **inPaths**: a map of paths and corresponding aliases.
- **outStream**: the standard ostream where to output the archive file.

void compress(const std::vector< tstring >& inPaths, const tstring& outFile) const

Compresses the given files or directories. The items in the first argument must be the relative or absolute paths to files or directories existing on the filesystem.

Parameters:

- **inPaths**: a vector of paths.
- **outFile**: the path (relative or absolute) to the output archive file.

void compress(const std::vector< tstring >& inPaths, std::ostream& outStream) const

Compresses the given files or directories. The items in the first argument must be the relative or absolute paths to files or directories existing on the filesystem.

Parameters:

- **inPaths**: a vector of paths.
- **outStream**: the standard ostream where the archive will be output.

void compressDirectory(const tstring& inDir, const tstring& outFile) const

Compresses an entire directory.

> **Note**
>
> This method is equivalent to compressFiles with filter set to L"".

Parameters:

- **inDir**: the path (relative or absolute) to the input directory.
- **outFile**: the path (relative or absolute) to the output archive file.

void compressDirectoryContents(const tstring& inDir, const tstring& outFile, bool recursive = true, const tstring& filter = "*") const

Compresses the contents of a directory.

> ⓘ Note
>
> Unlike compressFiles, this method includes also the metadata of the sub-folders.

Parameters:

- **inDir**: the path (relative or absolute) to the input directory.
- **outFile**: the path (relative or absolute) to the output archive file.
- **recursive**: (optional) if true, it searches the contents inside the sub-folders of inDir.
- **filter**: (optional) the filter to use when searching the contents inside inDir.

void compressFile(const tstring& inFile, const tstring& outFile, const tstring& inputName = {}) const

Compresses a single file.

Parameters:

- **inFile**: the file to be compressed.
- **outFile**: the path (relative or absolute) to the output archive file.
- **inputName**: (optional) the name to give to the compressed file inside the output archive.

void compressFile(const tstring& inFile, ostream& outStream, const tstring& inputName = {}) const

Compresses the input file to the output stream.

Parameters:

- **inFile**: the file to be compressed.
- **outStream**: the output stream.

- **inputName**: (optional) the name to give to the compressed file inside the output archive.

void compressFile(const tstring& inFile, vector< byte_t >& outBuffer, const tstring& inputName = {}) const

Compresses the input file to the output buffer.

Parameters:

- **inFile**: the file to be compressed.
- **outBuffer**: the buffer going to contain the output archive.
- **inputName**: (optional) the name to give to the compressed file inside the output archive.

void compressFiles(const std::vector< tstring >& inFiles, const tstring& outFile) const

Compresses a group of files.

> **Note**
>
> Any path to a directory or to a not-existing file will be ignored!

Parameters:

- **inFiles**: the path (relative or absolute) to the input files.
- **outFile**: the path (relative or absolute) to the output archive file.

void compressFiles(const tstring& inDir, const tstring& outFile, bool recursive = true, const tstring& filter = "*") const

Compresses the files contained in a directory.

Parameters:

- **inDir**: the path (relative or absolute) to the input directory.
- **outFile**: the path (relative or absolute) to the output archive file.
- **recursive**: (optional) if true, it searches files inside the sub-folders of inDir.
- **filter**: (optional) the filter to use when searching files inside inDir.

const BitInOutFormat & compressionFormat() const noexcept

Returns the format used for creating/updating an archive.

BitCompressionLevel compressionLevel() const noexcept

Returns the compression level used for creating/updating an archive.

BitCompressionMethod compressionMethod() const noexcept

Returns the compression method used for creating/updating an archive.

bool cryptHeaders() const noexcept

Returns whether the creator crypts also the headers of archives or not.

uint32_t dictionarySize() const noexcept

Returns the dictionary size used for creating/updating an archive.

FileCallback fileCallback() const

Returns the current file callback.

[virtual] const BitInFormat &override format() const noexcept

Returns the format used for creating/updating an archive.

bool isPasswordDefined() const noexcept

Returns a boolean value indicating whether a password is defined or not.

const Bit7zLibrary & library() const noexcept

Returns the Bit7zLibrary object used by the handler.

OverwriteMode overwriteMode() const

Returns the current OverwriteMode.

tstring password() const

Returns the password used to open, extract, or encrypt the archive.

PasswordCallback passwordCallback() const

Returns the current password callback.

ProgressCallback progressCallback() const

Returns the current progress callback.

RatioCallback ratioCallback() const

Returns the current ratio callback.

bool retainDirectories() const noexcept

Returns a boolean value indicating whether the directory structure must be preserved while extracting or compressing the archive.

void setCompressionLevel(BitCompressionLevel level) noexcept

Sets the compression level to be used when creating/updating an archive.

Parameters:

- **level**: the compression level desired.

void setCompressionMethod(BitCompressionMethod method)

Sets the compression method to be used when creating/updating an archive.

Parameters:

- **method**: the compression method desired.

void setDictionarySize(uint32_t dictionarySize)

Sets the dictionary size to be used when creating/updating an archive.

Parameters:

- **dictionarySize**: the dictionary size desired.

void setFileCallback(const FileCallback& callback)

Sets the function to be called when the current file being processed changes.

Parameters:

- **callback**: the file callback to be used.

void setFormatProperty(const wchar_t(&) name, const T& value) noexcept

Sets a property for the output archive format as described by the 7-zip documentation (e.g., https://sevenzip.osdn.jp/chm/cmdline/switches/method.htm). For example, passing the string L"tm" with a false value while creating a .7z archive will disable storing the last modified timestamps of the compressed files.

Parameters:

- **name**: The string name of the property to be set.
- **value**: The value to be used for the property.

void setFormatProperty(const wchar_t(&) name, T value) noexcept

Sets a property for the output archive format as described by the 7-zip documentation (e.g., https://sevenzip.osdn.jp/chm/cmdline/switches/method.htm).

Parameters:

- **name**: The string name of the property to be set.
- **value**: The value to be used for the property.

void setOverwriteMode(OverwriteMode mode)

Sets how the handler should behave when it tries to output to an existing file or buffer.

Parameters:

- **mode**: the OverwriteMode to be used by the handler.

[virtual] void setPassword(const tstring& password) override

Sets up a password for the output archives. When setting a password, the produced archives will be encrypted using the default cryptographic method of the output format. The option "crypt headers" remains unchanged, in contrast with what happens when calling the setPassword(tstring, bool) method.

> ⓘ Note
>
> Calling setPassword when the output format doesn't support archive encryption (e.g., GZip, BZip2, etc...) does not have any effects (in other words, it doesn't throw exceptions, and it has no effects on compression operations).

> ⓘ Note
>
> After a password has been set, it will be used for every subsequent operation. To disable the use of the password, you need to call the clearPassword method (inherited from BitAbstractArchiveHandler), which is equivalent to setPassword(L"").

Parameters:

- **password**: the password to be used when creating/updating archives.

void setPassword(const tstring& password, bool cryptHeaders)

Sets up a password for the output archive. When setting a password, the produced archive will be encrypted using the default cryptographic method of the output format. If the format is 7z, and the option "cryptHeaders" is set to true, the headers of the archive will be encrypted, resulting in a password request every time the output file will be opened.

> ⓘ Note
>
> Calling setPassword when the output format doesn't support archive encryption (e.g., GZip, BZip2, etc...) does not have any effects (in other words, it doesn't throw exceptions, and it has no effects on compression operations).

> ⓘ Note
>
> Calling setPassword with "cryptHeaders" set to true does not have effects on formats different from 7z.

> ⓘ Note
>
> After a password has been set, it will be used for every subsequent operation. To disable the use of the password, you need to call the clearPassword method (inherited from BitAbstractArchiveHandler), which is equivalent to setPassword(L"").

Parameters:

- **password**: the password to be used when creating/updating archives.
- **cryptHeaders**: if true, the headers of the output archives will be encrypted (valid only when using the 7z format).

void setPasswordCallback(const PasswordCallback& callback)

Sets the function to be called when a password is needed to complete the ongoing operation.

Parameters:

- **callback**: the password callback to be used.

void setProgressCallback(const ProgressCallback& callback)

Sets the function to be called when the processed size of the ongoing operation is updated.

> ⓘ Note
>
> The completion percentage of the current operation can be obtained by calculating `static_cast<int>((100.0 * processed_size) / total_size)`.

Parameters:

- **callback**: the progress callback to be used.

void setRatioCallback(const RatioCallback& callback)

Sets the function to be called when the input processed size and current output size of the ongoing operation are known.

> ⓘ Note
>
> The ratio percentage of a compression operation can be obtained by calculating `static_cast<int>((100.0 * output_size) / input_size)` .

Parameters:

- **callback**: the ratio callback to be used.

void setRetainDirectories(bool retain) noexcept

Sets whether the operations' output will preserve the input's directory structure or not.

Parameters:

- **retain**: the setting for preserving or not the input directory structure

void setSolidMode(bool solidMode) noexcept

Sets whether to use solid compression or not.

> ⓘ Note
>
> Setting the solid compression mode to true has effect only when using the 7z format with multiple input files.

Parameters:

- **solidMode**: if true, it will be used the "solid compression" method.

void setStoreSymbolicLinks(bool storeSymlinks) noexcept

Sets whether the creator will store symbolic links as links in the output archive.

Parameters:

- **storeSymlinks**: if true, symbolic links will be stored as links.

void setThreadsCount(uint32_t threadsCount) noexcept

Sets the number of threads to be used when creating/updating an archive.

Parameters:

- **threadsCount**: the number of threads desired.

void setTotalCallback(const TotalCallback& callback)

Sets the function to be called when the total size of an operation is available.

Parameters:

- **callback**: the total callback to be used.

void setUpdateMode(bool canUpdate)

Sets whether the creator can update existing archives or not.

> ⚠ Warning
>
> Deprecated since v4.0; it is provided just for an easier transition from the old v3 API.

> ⓘ Note
>
> If set to false, a subsequent compression operation may throw an exception if it targets an existing archive.

Parameters:

- **canUpdate**: if true, compressing operations will update existing archives.

[virtual] void setUpdateMode(UpdateMode mode)

Sets whether and how the creator can update existing archives or not.

> ⓘ Note
>
> If set to UpdateMode::None, a subsequent compression operation may throw an exception if it targets an existing archive.

Parameters:

- **mode**: the desired update mode.

void setVolumeSize(uint64_t volumeSize) noexcept

Sets the volumeSize (in bytes) of the output archive volumes.

> ⓘ Note
>
> This setting has effects only when the destination archive is on the filesystem.

Parameters:

- **volumeSize**: The dimension of a volume.

void setWordSize(uint32_t wordSize)

Sets the word size to be used when creating/updating an archive.

Parameters:

- **wordSize**: the word size desired.

bool solidMode() const noexcept

Returns whether the archive creator uses solid compression or not.

bool storeSymbolicLinks() const noexcept

Returns whether the archive creator stores symbolic links as links in the output archive.

uint32_t threadsCount() const noexcept

Returns the number of threads used when creating/updating an archive (a 0 value means that it will use the 7-zip default value).

TotalCallback totalCallback() const

Returns the current total callback.

UpdateMode updateMode() const noexcept

Returns the update mode used when updating existing archives.

uint64_t volumeSize() const noexcept

Returns the volume size (in bytes) used when creating multi-volume archives (a 0 value means that all files are going in a single archive).

uint32_t wordSize() const noexcept

Returns the word size used for creating/updating an archive.

Clone this wiki locally

```
https://github.com/rikyoz/bit7z.wiki.git
```