BitMemExtractor

Jump to bottom

Oz edited this page on Sep 26, 2024 · 15 revisions

Alias for `BitExtractor< const std::vector< byte_t > & >`.

The BitMemExtractor alias allows extracting the content of in-memory archives.

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

List of all members

Public Members

Return type	Name
	BitExtractor(const Bit7zLibrary& lib, const BitInFormat& format = BitFormat::Auto)
void	clearPassword() noexcept
void	extract(const std::vector< byte_t& > inArchive, const tstring& outDir = {}) const
void	extract(const std::vector< byte_t& > inArchive, std::map< tstring, vector< byte_t > >& outMap) const
void	extract(const std::vector< byte_t& > inArchive, std::ostream& outStream, uint32_t index = 0) const
void	extract(const std::vector< byte_t& > inArchive, vector< byte_t >& outBuffer, uint32_t index = 0) const
const BitInFormat &	extractionFormat() const noexcept
void	extractItems(const std::vector< byte_t& > inArchive, const std::vector< uint32_t >& indices, const tstring& outDir = {}) const

Return type	Name
void	extractMatching(const std::vector< byte_t& > inArchive, const tstring& itemFilter, const tstring& outDir = {}, FilterPolicy policy = FilterPolicy::Include) const
void	extractMatching(const std::vector< byte_t& > inArchive, const tstring& itemFilter, vector< byte_t >& outBuffer, FilterPolicy policy = FilterPolicy::Include) const
void	extractMatchingRegex(const std::vector< byte_t& > inArchive, const tstring& regex, const tstring& outDir = {}, FilterPolicy policy = FilterPolicy::Include) const
void	extractMatchingRegex(const std::vector< byte_t& > inArchive, const tstring& regex, vector< byte_t >& outBuffer, FilterPolicy policy = FilterPolicy::Include) const
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	setFileCallback(const FileCallback& callback)
void	setOverwriteMode(OverwriteMode mode)
void	setPassword(const tstring& password)
void	setPasswordCallback(const PasswordCallback& callback)
void	setProgressCallback(const ProgressCallback& callback)
void	setRatioCallback(const RatioCallback& callback)
void	setRetainDirectories(bool retain) noexcept

Return type	Name
void	setTotalCallback(const TotalCallback& callback)
void	test(const std::vector< byte_t& > inArchive) const
TotalCallback	totalCallback() const

Member Function Documentation

BitExtractor(const Bit7zLibrary& lib, const BitInFormat& format = BitFormat::Auto)

Constructs a BitExtractor object. The Bit7zLibrary parameter is needed to have access to the functionalities of the 7z DLLs. On the contrary, the BitInFormat is required to know the format of the in_file archives.

> **Note**
>
> When bit7z is compiled using the BIT7Z_AUTO_FORMAT macro define, the format argument has the default value BitFormat::Auto (automatic format detection of the in_file archive). Otherwise, when BIT7Z_AUTO_FORMAT is not defined (i.e., no auto format detection available), the format argument must be specified.

Parameters:

- **lib**: the 7z library to use.
- **format**: the in_file 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 extract(const std::vector< byte_t& > inArchive, const tstring& outDir = {}) const

Extracts the given archive to the chosen directory.

Parameters:

- **inArchive**: the input archive to be extracted.
- **outDir**: the output directory where extracted files will be put.

void extract(const std::vector< byte_t& > inArchive, std::map< tstring, vector< byte_t > >& outMap) const

Extracts the content of the given archive into a map of memory buffers, where the keys are the paths of the files (inside the archive), and the values are their decompressed contents.

Parameters:

- **inArchive**: the input archive to be extracted.
- **outMap**: the output map.

void extract(const std::vector< byte_t& > inArchive, std::ostream& outStream, uint32_t index = 0) const

Extracts a file from the given archive to the output stream.

Parameters:

- **inArchive**: the input archive to extract from.
- **outStream**: the (binary) stream where the content of the extracted file will be put.
- **index**: the index of the file to be extracted from the archive.

void extract(const std::vector< byte_t& > inArchive, vector< byte_t >& outBuffer, uint32_t index = 0) const

Extracts a file from the given archive to the output buffer.

Parameters:

- **inArchive**: the input archive to extract from.
- **outBuffer**: the output buffer where the content of the extracted file will be put.
- **index**: the index of the file to be extracted from the archive.

const BitInFormat & extractionFormat() const noexcept

Returns the archive format used by the archive opener.

void extractItems(const std::vector< byte_t& > inArchive, const std::vector< uint32_t >& indices, const tstring& outDir = {}) const

Extracts the specified items from the given archive to the chosen directory.

Parameters:

- **inArchive**: the input archive to extract from.
- **indices**: the indices of the files in the archive that should be extracted.
- **outDir**: the output directory where the extracted files will be placed.

void extractMatching(const std::vector< byte_t& > inArchive, const tstring& itemFilter, const tstring& outDir = {}, FilterPolicy policy = FilterPolicy::Include) const

Extracts the files in the archive that match the given wildcard pattern to the chosen directory.

Parameters:

- **inArchive**: the input archive to extract from.
- **itemFilter**: the wildcard pattern used for matching the paths of files inside the archive.
- **outDir**: the output directory where extracted files will be put.
- **policy**: the filtering policy to be applied to the matched items.

void extractMatching(const std::vector< byte_t& > inArchive, const tstring& itemFilter, vector< byte_t >& outBuffer, FilterPolicy policy = FilterPolicy::Include) const

Extracts to the output buffer the first file in the archive matching the given wildcard pattern.

Parameters:

- **inArchive**: the input archive to extract from.
- **itemFilter**: the wildcard pattern used for matching the paths of files inside the archive.
- **outBuffer**: the output buffer where to extract the file.
- **policy**: the filtering policy to be applied to the matched items.

void extractMatchingRegex(const std::vector< byte_t& > inArchive, const tstring& regex, const tstring& outDir = {}, FilterPolicy policy = FilterPolicy::Include) const

Extracts the files in the archive that match the given regex pattern to the chosen directory.

> **Note**
>
> Available only when compiling bit7z using the BIT7Z_REGEX_MATCHING preprocessor define.

Parameters:

- **inArchive**: the input archive to extract from.
- **regex**: the regex used for matching the paths of files inside the archive.
- **outDir**: the output directory where extracted files will be put.
- **policy**: the filtering policy to be applied to the matched items.

void extractMatchingRegex(const std::vector< byte_t& > inArchive, const tstring& regex, vector< byte_t >& outBuffer, FilterPolicy policy = FilterPolicy::Include) const

Extracts the first file in the archive that matches the given regex pattern to the output buffer.

> **Note**
>
> Available only when compiling bit7z using the BIT7Z_REGEX_MATCHING preprocessor define.

Parameters:

- **inArchive**: the input archive to extract from.
- **regex**: the regex used for matching the paths of files inside the archive.
- **outBuffer**: the output buffer where the extracted file will be put.
- **policy**: the filtering policy to be applied to the matched items.

FileCallback fileCallback() const

Returns the current file callback.

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

Returns the archive format used by the archive opener.

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 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 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)

Sets up a password to be used by the archive handler. The password will be used to encrypt/decrypt archives by using the default cryptographic method of the archive format.

> **Note**
>
> Calling setPassword when the input archive is not encrypted does not have any effect on the extraction process.

> **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, which is equivalent to calling setPassword(L"").

Parameters:

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

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 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 test(const std::vector< byte_t& > inArchive) const

Tests the given archive without extracting its content. If the archive is not valid, a BitException is thrown!

Parameters:

- **inArchive**: the input archive to be tested.

TotalCallback totalCallback() const

Returns the current total callback.