BitArchiveEditor

Jump to bottom

Oz edited this page on Sep 26, 2024 · 10 revisions

The BitArchiveEditor class allows creating new file archives or updating old ones. Update operations supported are the addition of new items, as well as renaming/updating/deleting old items;

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

*inherits from **BitArchiveWriter**.*

List of all members

Public Members

Return type	Name
	BitArchiveEditor(const Bit7zLibrary& lib, const tstring& inFile, const BitInOutFormat& format, const tstring& password = {})
void	addDirectory(const tstring& inDir)
void	addDirectoryContents(const tstring& inDir, const tstring& filter, bool recursive)
void	addDirectoryContents(const tstring& inDir, const tstring& filter = "*", FilterPolicy policy = FilterPolicy::Include, bool recursive = true)
void	addFile(const std::vector< byte_t >& inBuffer, const tstring& name)
void	addFile(const tstring& inFile, const tstring& name = {})
void	addFile(std::istream& inStream, const tstring& name)
void	addFiles(const std::vector< tstring >& inFiles)

Return type	Name
void	addFiles(const tstring& inDir, const tstring& filter, bool recursive)
void	addFiles(const tstring& inDir, const tstring& filter = "*", FilterPolicy policy = FilterPolicy::Include, bool recursive = true)
void	addItems(const std::map< tstring, tstring >& inPaths)
void	addItems(const std::vector< tstring >& inPaths)
void	applyChanges()
void	clearPassword() noexcept
const BitInOutFormat &	compressionFormat() const noexcept
BitCompressionLevel	compressionLevel() const noexcept
BitCompressionMethod	compressionMethod() const noexcept
void	compressTo(const tstring& outFile)
void	compressTo(std::ostream& outStream)
void	compressTo(std::vector< byte_t >& outBuffer)
const BitAbstractArchiveCreator &	creator() const noexcept
bool	cryptHeaders() const noexcept
void	deleteItem(const tstring& itemPath, DeletePolicy policy = DeletePolicy::ItemOnly)
void	deleteItem(uint32_t index, DeletePolicy policy = DeletePolicy::ItemOnly)
uint32_t	dictionarySize() const noexcept
FileCallback	fileCallback() const
const BitInFormat &override	format() const noexcept
const BitAbstractArchiveHandler &	handler() const noexcept
bool	isPasswordDefined() const noexcept
uint32_t	itemsCount() const

Return type	Name
const Bit7zLibrary &	library() const noexcept
OverwriteMode	overwriteMode() const
tstring	password() const
PasswordCallback	passwordCallback() const
ProgressCallback	progressCallback() const
RatioCallback	ratioCallback() const
void	renameItem(const tstring& oldPath, const tstring& newPath)
void	renameItem(uint32_t index, const tstring& newPath)
bool	retainDirectories() const noexcept
void	setCompressionLevel(BitCompressionLevel level) noexcept
void	setCompressionMethod(BitCompressionMethod method)
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

Return type	Name
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) override
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
void	updateItem(const tstring& itemPath, const std::vector< byte_t >& inBuffer)
void	updateItem(const tstring& itemPath, const tstring& inFile)
void	updateItem(const tstring& itemPath, istream& inStream)
void	updateItem(uint32_t index, const std::vector< byte_t >& inBuffer)
void	updateItem(uint32_t index, const tstring& inFile)
void	updateItem(uint32_t index, std::istream& inStream)
UpdateMode	updateMode() const noexcept
uint64_t	volumeSize() const noexcept
uint32_t	wordSize() const noexcept

Member Function Documentation

BitArchiveEditor(const Bit7zLibrary& lib, const tstring& inFile, const BitInOutFormat& format, const tstring& password = {})

Constructs a BitArchiveEditor object, reading the given archive file path.

Parameters:

- **lib**: the 7z library to use.
- **inFile**: the path to an input archive file.
- **format**: the input/output archive format.
- **password**: (optional) the password needed to read the input archive.

void addDirectory(const tstring& inDir)

Adds the given directory path and all its content.

Parameters:

- **inDir**: the path of the directory to be added to the archive.

void addDirectoryContents(const tstring& inDir, const tstring& filter, bool recursive)

Adds the contents of the given directory path. This function iterates through the specified directory and adds its contents based on the provided wildcard filter. Optionally, the operation can be recursive, meaning it will include subdirectories and their contents.

Parameters:

- **inDir**: the directory where to search for files to be added to the output archive.
- **filter**: the wildcard filter to be used for searching the files.
- **recursive**: recursively search the files in the given directory and all of its subdirectories.

void addDirectoryContents(const tstring& inDir, const tstring& filter = "*", FilterPolicy policy = FilterPolicy::Include, bool recursive = true)

Adds the contents of the given directory path. This function iterates through the specified directory and adds its contents based on the provided wildcard filter and policy. Optionally, the operation can be recursive, meaning it will include subdirectories and their contents.

Parameters:

- **inDir**: the directory where to search for files to be added to the output archive.
- **filter**: (optional) the wildcard filter to be used for searching the files.

- **recursive**: (optional) recursively search the files in the given directory and all of its subdirectories.
- **policy**: (optional) the filtering policy to be applied to the matched items.

void addFile(const std::vector< byte_t >& inBuffer, const tstring& name)

Adds the given buffer file, using the given name as a path when compressed in the output archive.

Parameters:

- **inBuffer**: the buffer containing the file to be added to the output archive.
- **name**: user-defined path to be used inside the output archive.

void addFile(const tstring& inFile, const tstring& name = {})

Adds the given file path, with an optional user-defined path to be used in the output archive.

> **Note**
>
> If a directory path is given, a BitException is thrown.

Parameters:

- **inFile**: the path to the filesystem file to be added to the output archive.
- **name**: (optional) user-defined path to be used inside the output archive.

void addFile(std::istream& inStream, const tstring& name)

Adds the given standard input stream, using the given name as a path when compressed in the output archive.

Parameters:

- **inStream**: the input stream to be added.
- **name**: the name of the file inside the output archive.

void addFiles(const std::vector< tstring >& inFiles)

Adds all the files in the given vector of filesystem paths.

> **Note**
>
> Paths to directories are ignored.

Parameters:

- **inFiles**: the vector of paths to files.

void addFiles(const tstring& inDir, const tstring& filter, bool recursive)

Adds all the files inside the given directory path that match the given wildcard filter.

Parameters:

- **inDir**: the directory where to search for files to be added to the output archive.
- **filter**: the wildcard filter to be used for searching the files.
- **recursive**: recursively search the files in the given directory and all of its subdirectories.

void addFiles(const tstring& inDir, const tstring& filter = "*", FilterPolicy policy = FilterPolicy::Include, bool recursive = true)

Adds all the files inside the given directory path that match the given wildcard filter.

Parameters:

- **inDir**: the directory where to search for files to be added to the output archive.
- **filter**: (optional) the wildcard filter to be used for searching the files.
- **recursive**: (optional) recursively search the files in the given directory and all of its subdirectories.
- **policy**: (optional) the filtering policy to be applied to the matched items.

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

Adds all the items that can be found by indexing the keys of the given map of filesystem paths; the corresponding mapped values are the user-defined paths wanted inside the output archive.

Parameters:

- **inPaths**: map of filesystem paths with the corresponding user-defined path desired inside the output archive.

void addItems(const std::vector< tstring >& inPaths)

Adds all the items that can be found by indexing the given vector of filesystem paths.

Parameters:

- **inPaths**: the vector of filesystem paths.

void applyChanges()

Applies the requested changes (i.e., rename/update/delete operations) to the input archive.

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"").

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.

void compressTo(const tstring& outFile)

Compresses all the items added to this object to the specified archive file path.

> **Note**

> If this object was created by passing an input archive file path, and this latter is the same as the outFile path parameter, the file will be updated.

Parameters:

- **outFile**: the output archive file path.

void compressTo(std::ostream& outStream)

Compresses all the items added to this object to the specified buffer.

Parameters:

- **outStream**: the output standard stream.

void compressTo(std::vector< byte_t >& outBuffer)

Compresses all the items added to this object to the specified buffer.

Parameters:

- **outBuffer**: the output buffer.

const BitAbstractArchiveCreator & creator() const noexcept

Returns a constant reference to the BitAbstractArchiveHandler object containing the settings for writing the output archive.

bool cryptHeaders() const noexcept

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

void deleteItem(const tstring& itemPath, DeletePolicy policy = DeletePolicy::ItemOnly)

Marks as deleted the archive's item(s) with the specified path.

> **Note**

> By default, if the marked item is a folder, only its metadata will be deleted, not the files within it. To delete the folder contents as well, set the `policy` to `DeletePolicy::RecurseDirs`.

> **Note**
>
> The specified path must not begin with a path separator.

> **Note**
>
> A path with a trailing separator will *only* be considered if the policy is DeletePolicy::RecurseDirs, and will only match folders; with DeletePolicy::ItemOnly, no item will match a path with a trailing separator.

> **Note**
>
> Generally, archives may contain multiple items with the same paths. If this is the case, all matching items will be marked as deleted according to the specified policy.

Parameters:

- **itemPath**: the path (in the archive) of the item to be deleted.
- **policy**: the policy to be used when deleting items.

> **Important**
>
> The function throws a BitException if the specified path is empty or invalid, or if no matching item could be found.

void deleteItem(uint32_t index, DeletePolicy policy = DeletePolicy::ItemOnly)

Marks as deleted the item at the given index.

> **Note**
>
> By default, if the item is a folder, only its metadata is deleted, not the files within it. If instead the policy is set to DeletePolicy::RecurseDirs, then the items within the folder will also be deleted.

Parameters:

- **index**: the index of the item to be deleted.
- **policy**: the policy to be used when deleting items.

> **Important**
>
> The function throws a BitException if the index is invalid.

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.

const BitAbstractArchiveHandler & handler() const noexcept

Returns a constant reference to the BitAbstractArchiveHandler object containing the settings for writing the output archive.

bool isPasswordDefined() const noexcept

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

uint32_t itemsCount() const

Returns the total number of items added to the output archive object.

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.

void renameItem(const tstring& oldPath, const tstring& newPath)

Requests to change the path of the item from oldPath to the newPath.

Parameters:

- **oldPath**: the old path (in the archive) of the item to be renamed.
- **newPath**: the new path (in the archive) desired for the item.

void renameItem(uint32_t index, const tstring& newPath)

Requests to change the path of the item at the specified index with the given one.

Parameters:

- **index**: the index of the item to be renamed.
- **newPath**: the new path (in the archive) desired for the item.

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) override

Sets how the editor performs the update of the items in the archive.

> **Note**
>
> BitArchiveEditor doesn't support UpdateMode::None.

Parameters:

- **mode**: the desired update mode (either UpdateMode::Append or UpdateMode::Overwrite).

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.

void updateItem(const tstring& itemPath, const std::vector< byte_t >& inBuffer)

Requests to update the content of the item at the specified path with the data from the given buffer.

Parameters:

- **itemPath**: the path (in the archive) of the item to be updated.
- **inBuffer**: the buffer containing the new data for the item.

void updateItem(const tstring& itemPath, const tstring& inFile)

Requests to update the content of the item at the specified path with the data from the given file.

Parameters:

- **itemPath**: the path (in the archive) of the item to be updated.
- **inFile**: the path to the file containing the new data for the item.

void updateItem(const tstring& itemPath, istream& inStream)

Requests to update the content of the item at the specified path with the data from the given stream.

Parameters:

- **itemPath**: the path (in the archive) of the item to be updated.
- **inStream**: the stream of new data for the item.

void updateItem(uint32_t index, const std::vector< byte_t >& inBuffer)

Requests to update the content of the item at the specified index with the data from the given buffer.

Parameters:

- **index**: the index of the item to be updated.
- **inBuffer**: the buffer containing the new data for the item.

void updateItem(uint32_t index, const tstring& inFile)

Requests to update the content of the item at the specified index with the data from the given file.

Parameters:

- **index**: the index of the item to be updated.
- **inFile**: the path to the file containing the new data for the item.

void updateItem(uint32_t index, std::istream& inStream)

Requests to update the content of the item at the specified index with the data from the given stream.

Parameters:

- **index**: the index of the item to be updated.
- **inStream**: the stream of new data for the item.

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.