BitArchiveReader

Jump to bottom

Oz edited this page on Dec 16, 2024 · 9 revisions

The BitArchiveReader class allows reading metadata of archives, as well as extracting them.

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

*inherits from **BitAbstractArchiveOpener**, **BitInputArchive**.*

List of all members

Public Members

Return type	Name
	BitArchiveReader(const Bit7zLibrary& lib, const buffer_t& inArchive, ArchiveStartOffset archiveStart, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})
	BitArchiveReader(const Bit7zLibrary& lib, const std::vector< byte_t >& inArchive, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})
	BitArchiveReader(const Bit7zLibrary& lib, const tstring& inArchive, ArchiveStartOffset archiveStart, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})
	BitArchiveReader(const Bit7zLibrary& lib, const tstring& inArchive, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})
	BitArchiveReader(const Bit7zLibrary& lib, std::istream& inArchive, ArchiveStartOffset

Return type	Name
	archiveStart, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})
	BitArchiveReader(const Bit7zLibrary& lib, std::istream& inArchive, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})
	~BitArchiveReader() override = default
const tstring &	archivePath() const noexcept
map< BitProperty, BitPropVariant >	archiveProperties() const
BitPropVariant	archiveProperty(BitProperty property) const
BitInputArchive::ConstIterator	begin() const noexcept
BitInputArchive::ConstIterator	cbegin() const noexcept
BitInputArchive::ConstIterator	cend() const noexcept
void	clearPassword() noexcept
bool	contains(const tstring& path) const noexcept
const BitInFormat &	detectedFormat() const noexcept
BitInputArchive::ConstIterator	end() const noexcept
const BitInFormat &	extractionFormat() const noexcept
void	extractTo(byte_t* buffer, std::size_t size, uint32_t index = 0) const
void	extractTo(byte_t(&buffer)[N], uint32_t index = 0) const
void	extractTo(const tstring& outDir) const
void	extractTo(const tstring& outDir, const std::vector< uint32_t >& indices) const
void	extractTo(std::array< byte_t, N >& buffer, uint32_t index = 0) const
void	extractTo(std::map< tstring, std::vector< byte_t > >& outMap) const
void	extractTo(std::ostream& outStream, uint32_t index = 0) const

Return type	Name
void	extractTo(std::vector< byte_t >& outBuffer, uint32_t index = 0) const
FileCallback	fileCallback() const
uint32_t	filesCount() const
BitInputArchive::ConstIterator	find(const tstring& path) const noexcept
uint32_t	foldersCount() const
const BitInFormat &override	format() const noexcept
const BitAbstractArchiveHandler &	handler() const noexcept
bool	hasEncryptedItems() const
bool	isEncrypted() const
bool	isItemEncrypted(uint32_t index) const
bool	isItemFolder(uint32_t index) const
bool	isMultiVolume() const
bool	isPasswordDefined() const noexcept
bool	isSolid() const
BitArchiveItemOffset	itemAt(uint32_t index) const
BitPropVariant	itemProperty(uint32_t index, BitProperty property) const
vector< BitArchiveItemInfo >	items() const
uint32_t	itemsCount() const
const Bit7zLibrary &	library() const noexcept
OverwriteMode	overwriteMode() const
uint64_t	packSize() const
tstring	password() const
PasswordCallback	passwordCallback() const
ProgressCallback	progressCallback() const
RatioCallback	ratioCallback() const

Return type	Name
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
void	setTotalCallback(const TotalCallback& callback)
uint64_t	size() const
void	test() const
void	testItem(uint32_t index) const
TotalCallback	totalCallback() const
void	useFormatProperty(const wchar_t* name, const BitPropVariant& property) const
void	useFormatProperty(const wchar_t* name, T&& value) const
uint32_t	volumesCount() const

Static Public Members

Return type	Name
bool	isEncrypted(const Bit7zLibrary& lib, T&& inArchive, const BitInFormat& format = BitFormat::Auto)
bool	isHeaderEncrypted(const Bit7zLibrary& lib, T&& inArchive, const BitInFormat& format = BitFormat::Auto)

Member Function Documentation

BitArchiveReader(const Bit7zLibrary& lib, const buffer_t& inArchive, ArchiveStartOffset archiveStart, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})

Constructs a BitArchiveReader object, opening the archive in the input buffer.

> **Note**
>
> When bit7z is compiled using the `BIT7Z_AUTO_FORMAT` option, the format argument has the default value BitFormat::Auto (automatic format detection of the input archive). On the contrary, 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 used.
- **inArchive**: the input buffer containing the archive to be read.
- **archiveStart**: whether to search for the archive's start throughout the entire file or only at the beginning.
- **format**: the format of the input archive.
- **password**: (optional) the password needed for opening the input archive.

BitArchiveReader(const Bit7zLibrary& lib, const std::vector< byte_t >& inArchive, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})

Constructs a BitArchiveReader object, opening the archive in the input buffer.

> **Note**
>
> When bit7z is compiled using the `BIT7Z_AUTO_FORMAT` option, the format argument has the default value BitFormat::Auto (automatic format detection of the input archive). On the contrary, 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 used.
- **inArchive**: the input buffer containing the archive to be read.
- **format**: the format of the input archive.
- **password**: (optional) the password needed for opening the input archive.

BitArchiveReader(const Bit7zLibrary& lib, const tstring& inArchive, ArchiveStartOffset archiveStart, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})

Constructs a BitArchiveReader object, opening the input file archive.

> **Note**
>
> When bit7z is compiled using the `BIT7Z_AUTO_FORMAT` option, the format argument has the default value BitFormat::Auto (automatic format detection of the input archive). On the contrary, 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 used.
- **inArchive**: the path to the archive to be read.
- **archiveStart**: whether to search for the archive's start throughout the entire file or only at the beginning.
- **format**: the format of the input archive.
- **password**: (optional) the password needed for opening the input archive.

BitArchiveReader(const Bit7zLibrary& lib, const tstring& inArchive, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})

Constructs a BitArchiveReader object, opening the input file archive.

> **Note**
>
> When bit7z is compiled using the `BIT7Z_AUTO_FORMAT` option, the format argument has the default value BitFormat::Auto (automatic format detection of the input archive). On the contrary, 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 used.
- **inArchive**: the path to the archive to be read.
- **format**: the format of the input archive.
- **password**: (optional) the password needed for opening the input archive.

BitArchiveReader(const Bit7zLibrary& lib, std::istream& inArchive, ArchiveStartOffset archiveStart, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})

Constructs a BitArchiveReader object, opening the archive from the standard input stream.

> **Note**
>
> When bit7z is compiled using the `BIT7Z_AUTO_FORMAT` option, the format argument has the default value BitFormat::Auto (automatic format detection of the input archive). On the contrary, 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 used.
- **inArchive**: the standard input stream of the archive to be read.
- **archiveStart**: whether to search for the archive's start throughout the entire file or only at the beginning.
- **format**: the format of the input archive.
- **password**: (optional) the password needed for opening the input archive.

BitArchiveReader(const Bit7zLibrary& lib, std::istream& inArchive, const BitInFormat& format = BitFormat::Auto, const tstring& password = {})

Constructs a BitArchiveReader object, opening the archive from the standard input stream.

> **Note**
>
> When bit7z is compiled using the `BIT7Z_AUTO_FORMAT` option, the format argument has the default value BitFormat::Auto (automatic format detection of the input archive). On the contrary, 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 used.
- **inArchive**: the standard input stream of the archive to be read.
- **format**: the format of the input archive.
- **password**: (optional) the password needed for opening the input archive.

~BitArchiveReader() override = default

BitArchiveReader destructor.

> **Note**
>
> It releases the input archive file.

const tstring & archivePath() const noexcept

Returns the path to the archive (the empty string for buffer/stream archives).

map< BitProperty, BitPropVariant > archiveProperties() const

Returns a map of all the available (i.e., non-empty) archive properties and their respective values.

BitPropVariant archiveProperty(BitProperty property) const

Gets the specified archive property.

Parameters:

- **property**: the property to be retrieved.

Returns the current value of the archive property or an empty BitPropVariant if no value is specified.

BitInputArchive::ConstIterator begin() const noexcept

Returns an iterator to the first element of the archive; if the archive is empty, the returned iterator will be equal to the end() iterator.

BitInputArchive::ConstIterator cbegin() const noexcept

Returns an iterator to the first element of the archive; if the archive is empty, the returned iterator will be equal to the end() iterator.

BitInputArchive::ConstIterator cend() const noexcept

Returns an iterator to the element following the last element of the archive; this element acts as a placeholder: attempting to access it results in undefined behavior.

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"").

bool contains(const tstring& path) const noexcept

Find if there is an item in the archive that has the given path.

Parameters:

- **path**: the path to be searched in the archive.

Returns true if and only if an item with the given path exists in the archive.

const BitInFormat & detectedFormat() const noexcept

Returns the detected format of the file.

BitInputArchive::ConstIterator end() const noexcept

Returns an iterator to the element following the last element of the archive; this element acts as a placeholder: attempting to access it results in undefined behavior.

const BitInFormat & extractionFormat() const noexcept

Returns the archive format used by the archive opener.

void extractTo(byte_t* buffer, std::size_t size, uint32_t index = 0) const

Extracts a file to the pre-allocated output buffer.

Parameters:

- **buffer**: the pre-allocated output buffer.
- **size**: the size of the output buffer (it must be equal to the unpacked size of the item to be extracted).
- **index**: the index of the file to be extracted.

void extractTo(byte_t(&) buffer, uint32_t index = 0) const

Extracts a file to the pre-allocated output buffer.

Parameters:

- **buffer**: the pre-allocated output buffer.
- **index**: the index of the file to be extracted.

void extractTo(const tstring& outDir) const

Extracts the archive to the chosen directory.

Parameters:

- **outDir**: the output directory where the extracted files will be put.

void extractTo(const tstring& outDir, const std::vector< uint32_t >& indices) const

Extracts the specified items to the chosen directory.

Parameters:

- **outDir**: the output directory where the extracted files will be put.
- **indices**: the array of indices of the files in the archive that must be extracted.

void extractTo(std::array< byte_t, N >& buffer, uint32_t index = 0) const

Extracts a file to the pre-allocated output buffer.

Parameters:

- **buffer**: the pre-allocated output buffer.
- **index**: the index of the file to be extracted.

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

Extracts the content of the archive to 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:

- **outMap**: the output map.

void extractTo(std::ostream& outStream, uint32_t index = 0) const

Extracts a file to the output stream.

Parameters:

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

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

Extracts a file to the output buffer.

Parameters:

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

FileCallback fileCallback() const

Returns the current file callback.

uint32_t filesCount() const

Returns the number of files contained in the archive.

BitInputArchive::ConstIterator find(const tstring& path) const noexcept

Find an item in the archive that has the given path.

Parameters:

- **path**: the path to be searched in the archive.

Returns an iterator to the item with the given path, or an iterator equal to the end() iterator if no item is found.

uint32_t foldersCount() const

Returns the number of folders contained in the archive.

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

Returns the archive format used by the archive opener.

const BitAbstractArchiveHandler & handler() const noexcept

Returns the BitAbstractArchiveHandler object containing the settings for reading the archive.

bool hasEncryptedItems() const

Returns true if and only if the archive has at least one encrypted item.

bool isEncrypted() const

Returns true if and only if the archive has only encrypted items.

bool isItemEncrypted(uint32_t index) const

Parameters:

- **index**: the index of an item in the archive.

Returns true if and only if the item at the given index is encrypted.

bool isItemFolder(uint32_t index) const

Parameters:

- **index**: the index of an item in the archive.

Returns true if and only if the item at the given index is a folder.

bool isMultiVolume() const

Returns true if and only if the archive is composed by multiple volumes.

bool isPasswordDefined() const noexcept

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

bool isSolid() const

Returns true if and only if the archive was created using solid compression.

BitArchiveItemOffset itemAt(uint32_t index) const

Retrieve the item at the given index.

Parameters:

- **index**: the index of the item to be retrieved.

Returns the item at the given index within the archive.

BitPropVariant itemProperty(uint32_t index, BitProperty property) const

Gets the specified property of an item in the archive.

Parameters:

- **index**: the index (in the archive) of the item.
- **property**: the property to be retrieved.

Returns the current value of the item property or an empty BitPropVariant if the item has no value for the property.

vector< BitArchiveItemInfo > items() const

Returns a vector of all the archive items as BitArchiveItem objects.

uint32_t itemsCount() const

Returns the number of items contained in the archive.

const Bit7zLibrary & library() const noexcept

Returns the Bit7zLibrary object used by the handler.

OverwriteMode overwriteMode() const

Returns the current OverwriteMode.

uint64_t packSize() const

Returns the total compressed size of the archive content.

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.

uint64_t size() const

Returns the total uncompressed size of the archive content.

void test() const

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

void testItem(uint32_t index) const

Tests the item at the given index inside the archive without extracting it. If the archive is not valid, or there's no item at the given index, a BitException is thrown!

Parameters:

- **index**: the index of the file to be tested.

TotalCallback totalCallback() const

Returns the current total callback.

void useFormatProperty(const wchar_t* name, const BitPropVariant& property) const

Use the given format property to read the archive.

Parameters:

- **name**: the name of the property.
- **property**: the property value.

void useFormatProperty(const wchar_t* name, T&& value) const

Use the given format property to read the archive.

Parameters:

- **name**: the name of the property.
- **value**: the property value.

uint32_t volumesCount() const

Returns the number of volumes composing the archive.

[static] bool isEncrypted(const Bit7zLibrary& lib, T&& inArchive, const BitInFormat& format = BitFormat::Auto)

Checks if the given archive contains only encrypted items.

> **Note**
>
> A header-encrypted archive is also encrypted, but the contrary is not generally true.

> **Note**
>
> An archive might contain both plain and encrypted files; in this case, this function will return false.

Parameters:

- **lib**: the 7z library used.
- **inArchive**: the archive to be read.
- **format**: the format of the input archive.

Returns true if and only if the archive has only encrypted items.

[static] bool isHeaderEncrypted(const Bit7zLibrary& lib, T&& inArchive, const BitInFormat& format = BitFormat::Auto)

Checks if the given archive is header-encrypted or not.

Parameters:

- **lib**: the 7z library used.

- **inArchive**: the archive to be read.
- **format**: the format of the input archive.

Returns true if and only if the archive has at least one encrypted item.

bit7z