Zip / gzip Multiplatform Native Plugin.

Thank you for purchasing this shared library for Android, iOS*, Windows, OSX, Linux & WebGL**.
The scope of this library is to compress/decompress zip/gzip/tar/bz2 archives and buffers on Android, iOS, tvOS, Windows, OSX, Linux & WebGL.

The iOS libraries are compiled for arm64 and non bitcode. Bitcode binaries are provided. There are also binaries compiled for arm64 and armv7 support if needed.

(bitcode enabled iOS plugins are provided in the plugins/ios/bitcode folder. **If you are creating a bitcode enabled project please use these provided plugins!**)

OSX bundle is compiled now as universal only(**Intel/arm64**). An **x86_64** version (Intel) and an **arm64** (Silicon) only version are included as zips.

If you are publishing for MacOS on the AppStore please do the following to codesign the bundle:

1. Remove all the meta files from inside the .bundle.
2. Remove (if any) folder named CodeSignature.
3. Find the Info.plist file and change the bundleID to one of your own.

The Windows and Linux libraries are compiled for x86 and x86_64 build modes.
The Android lib is compiled for armeabi, armeabi-v7a, x86, x86_64 and arm64-v8a. Minimum version api=16.
Armv7 and x86 include also plugins with minimum api=24 (7.0 Nougat) to support files larger than 2GB.

*** For Unity 2022.x+ the WebGL demo requires to enable http connections.**
**** webGL/tvOS support all functions except the ones that use the FileSystem.**
***** bz2 compression method is not available for WebGL and MacOS/iOS/tvOS.**

FEATURES:

Fast zip/gzip compression and decompression with a clean and simple interface. Very easy to use.
The plugin is about **7x times faster** in compression speed and **3x times faster** in decompression speed compared to SharpZipLib.

- compress/decompress buffers to/from zlib/gzip streams.
- recursive directory compression/decompression.
- compress/decompress single files.
- encryption / decryption.
- append files to existing zip archives.
- **compress a buffer and write it or append it to a zip archive.**
- get file and size info of all the files or a specific file from a zip archive.
- extract a single file out of a zip archive.
- extract a list of entries to the file system or in multiple buffers.
- decompress a file of a zip archive to a byte buffer.
- delete an entry in a zip archive.
- replace an entry in a zip archive.
- get progress of extraction when the zip archive has multiple files and progress of single file decompression.
- get **byte level progress** of compression/decompression (single or multiple files).
- create i**n Memory** zip files and manipulate them from there.
- create **split zip archives** (in the form of 1.zip, 1.z01, 1.z02, ...)
- decompression of **split zip archives (**in the form of 1.zip, 1.z01, 1.z02, …)
- ability to **discover** merged/hidden zip/gzip archives in files or buffers and extract from there.
- bz2/zlib compression-decompression methods. (bz2 method not available for MacOS/iOS/tvOS & webGL)
- ***buffers can be treated as files****. That means if you have a file in www.bytes you can perform operations directly on the buffer. (For Android this is very useful since you can decompress from Streaming Assets without copying to Persistent data path.)*

- support for ***native file buffers*** *to avoid memory spikes when decompressing from managed buffers.*
- *support for* ***STORE*** *method when setting level of compression to zero.*
- *ability to cancel most zip/unzip/gzip/tar operations when they are called from a Thread.*
- *create/decompress* ***gzip*** *files.*
- *decompress gzip from buffer to the file system.*
- *ability to create* ***tar*** *archives from directories or list of files with byte level progress.*
- *untar and untar entry support with byte level progress.*
- *compress/decompress of* ***tar.gz****,* ***tar.bz2*** *with progress.*
- *create and decompress* ***bz2*** *archives with progress.*

!!! The plugin will not decompress __MACOSX folders, .DS_Store files and files starting with ._ !!!

INSTRUCTIONS:

If you want to run a small example, compile and run the testScene.
It will download a small zip file and it will perform all the functions the lib provides.
See the **lzip.cs** file for more comments and error codes.
In your project include in the Plugins folder the plugins you want to use and the lzip.cs file
and call the appropriate functions
as described below and shown in the demo scene.

```
To get a progress percentage:

Use the lzip.getFileInfo function. This function returns the total uncompressed bytes of the
contents in the zip (among other things).
Then when decompressing (through a separate thread), divide the proc[0] returned by the function
with the total bytes returned from the lzip.getFileInfo function.
Like this (float)(totalBytes / proc[0]). This will go from 0.0f to 1.0f. Multiply this with 100 and
you will have the progress.

The same for compression. Only this time you have to make the sum of the total file sizes and
compare against the byteProgress of the compress_File/compressDir/compress_File_List function.
```

THE FUNCTIONS:

The public static lists are:

```
List<string> ninfo = new List<string>();//filenames
List<UInt64> uinfo = new List<UInt64>();//uncompressed file sizes
List<UInt64> cinfo = new List<UInt64>();//compressed file sizes
List<UInt64> localOffset = new List<UInt64>();//Local offset file header. (usefull when the STORE
                                                method is used to grab a file from the zip.)

int zipFiles, zipFolders;// global integers that store the number of files and folders in a zip
file.


int getEntryIndex(string entry);

This function returns the index of an entry assuming the getFileInfo function was called prior on a
zip file.

entry:       the entry for which we want to get the index.
Returns -1 if no entry was found in the List.
```

```
void setCancel();
```

Send cancel signal to the following operations:
- compress a single file
- compress directory or list of files
- extract a single file to the file system
- extract a zip to the file system.
- entry2Buffer
- bz2/tar/gz file operations

This function is useful when the zip operation is executed in a Thread.

```
int getTotalFiles(string zipArchive, object fileBuffer = null);
```

A function that returns the total number of files in a zip archive.

zipArchive : the zip to be checked
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!
ERROR CODES : -1 = failed to access zip archive
: -3 = non valid zip size when using native file buffer
: any number>0 = the number of files in the zip archive

```
int getTotalEntries(string zipArchive, object fileBuffer = null);
```

A function that will return the total entries in a zip archive. (files + folders)

zipArchive : the zip to be checked
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!
ERROR CODES : -2 = failed to access zip archive
: -3 = non valid zip size when using native file buffer
: any number>0 = the number of entries in the zip archive

```
UInt64 getFileInfo(string zipArchive, string path = null, object fileBuffer = null);
```

This function fills the Lists with the filenames and file sizes that are in the zip file ***(see page 12)***

Returns : the total size of uncompressed bytes of the files in the zip archive

zipArchive : the full path to the archive, including the archives name. (/myPath/myArchive.zip)
path : *this is no longer used. It is kept for a while for backwards* compatibility.
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!
ERROR CODES : 0 = Input file not found **or** could not get info **or** fileBuffer didn't input correct zip file size.

```
ulong getEntrySize(string zipArchive, string entry, object fileBuffer = null);
```

A function that returns the uncompressed size of a file in a zip archive.

zipArchive : the zip archive to get the info from.
entry : the entry for which we want to know it uncompressed size.
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!

```
bool entryExists(string zipArchive, string entry, object fileBuffer = null);
```

A function that tells if an entry in zip archive exists.

Returns true or false.
zipArchive : the zip archive to get the info from.
entry : the entry for which we want to know if it exists.
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!

```
bool compressBuffer(byte[] source, ref byte[] outBuffer, int levelOfCompression);
```

A function that compresses a byte buffer to a zlib stream compressed buffer. Provide a reference buffer to write to. This buffer will be resized.

source : the input buffer
outBuffer : the referenced output buffer
levelOfCompression : (0-10) recommended 9 for maximum (10 is highest but slower and not zlib compatible)
ERROR CODES : true = success
: false = failed

```
byte[] compressBuffer(byte[] source, int levelOfCompression);
```

A function that compresses a byte buffer to a zlib stream compressed buffer. Returns a new buffer with the compressed data.

source : the input buffer
levelOfCompression : (0-10) recommended 9 for maximum (10 is highest but slower and not zlib compatible)
ERROR CODES : a valid byte buffer = success
: null = failed

```
int compressBufferFixed(byte[] source, ref byte[] outBuffer, int levelOfCompression, bool safe = true);
```

Same as the compressBuffer function, only this function will put the result in a fixed size buffer to avoid memory allocations.
The compressed size is returned so you can manipulate it at will.

safe: if set to true the function will abort if the compressed resut is larger the the fixed size output buffer.
Otherwise compressed data will be written only until the end of the fixed output buffer.

```
bool decompressBuffer(byte[] source, ref byte[] outBuffer);
```

A function that decompresses a zlib compressed buffer to a referenced outBuffer. The outbuffer will be resized.

source : a zlib compressed buffer.
outBuffer : a referenced out buffer provided to extract the data. This buffer will be resized to fit the uncompressed data.
ERROR CODES : true = success
: false = failed

```
byte[] decompressBuffer(byte[] source);
```

A function that decompresses a zlib compressed buffer and creates a new buffer. Returns a new buffer with the uncompressed data.

source : a zlib compressed buffer.
ERROR CODES : a valid byte buffer = success
: null = failed

```
int decompressBufferFixed(byte[] source, ref byte[] outBuffer, bool safe = true);
```

Same as the decompressBuffer function. Only this one outputs to a buffer of fixed which size isn't resized to avoid memory allocations.

The fixed buffer should have a size that will be able to hold the incoming decompressed data.

Returns the uncompressed size.
safe: if set to true the function will abort if the decompressed resut is larger the the fixed size output buffer.
Otherwise decompressed data will be written only until the end of the fixed output buffer.

```
int entry2Buffer(string zipArchive, string entry, ref byte[] buffer, object fileBuffer = null, string password = null);
```

A function that will decompress a file in a zip archive directly in a provided byte buffer.

zipArchive : the full path to the zip archive from which a specific file will be extracted to a byte buffer.
entry : the file we want to extract to a buffer. (If the file resides in a directory, the directory should be included.)
buffer : a referenced byte buffer that will be resized and will be filled with the extraction data.
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!
password : If the archive is encrypted use a password.
ERROR CODES : 1 = success
: -2 = could not find/open zip archive
: -3 = could not locate entry

: -4 = could not get entry info
: -6 = non valid zip size when using native file buffer
: -5 = password error
: -104 = internal memory error

---

byte[] entry2Buffer(string zipArchive, string entry, object fileBuffer = null, string password = null));

A function that will decompress a file in a zip archive in a new created and returned byte buffer.

zipArchive : the full path to the zip archive from which a specific file will be extracted to a byte buffer.
entry : the file we want to extract to a buffer. (If the file resides in a directory, the directory should be included.)
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!
password : If the archive is encrypted use a password.
ERROR CODES : non-null = success
: null = failed

---

bool buffer2File(int levelOfCompression, string zipArchive, string arcFilename, byte[] buffer, bool append = false);

A function that compresses a byte buffer and writes it to a zip file. I you set the append flag to true, the output will get appended to an existing zip archive.

levelOfCompression : (0-9) recommended 9 for maximum. (0 = Store method.)
zipArchive : the full path to the zip archive to be created or append to.
arcFilename : the name of the file that will be written to the archive.
buffer : the buffer that will be compressed and will be put in the zip archive.
append : set true if you want the output to be appended to an existing zip archive.
comment : an optional comment for this entry.
password : an optional password to encrypt this entry.
useBz2 : set to true if you want bz2 compression instead of zlib. (Except MacOS/iOS/tvOS/watchOS/webGL)
ERROR CODES : true = success
: false = failed

---

int delete_entry(string zipArchive, string arcFilename);

A function that deletes a file in a zip archive. It creates a temp file where the compressed data of the old archive is copied except the one that needs to be deleted.
After that the old zip archive is deleted and the temp file gets renamed to the original zip archive.
You can delete directories too if they are empty. Does not work on password protected archives.

zipArchive : the full path to the zip archive
arcFilename : the name of the file that will be deleted.
ERROR CODES : 1 = success
: -1 = failed to open zip
: -2 = failed to locate the archive to be deleted in the zip file
: -3 = error copying compressed data from original zip
: -4 = failed to create temp zip file.

```
int replace_entry(string zipArchive, string arcFilename, string newFilePath, int level = 9, string comment = null, string password = null, bool useBz2 = false);
```

A function that replaces an entry in a zip archive with a file that lies in a path. The original name of the archive will be used. Does not work on password protected archives.

```
zipArchive     : the full path to the zip archive
arcFilename    : the name of the file that will be replaced.
newFilePath    : a path to the file that will replace the original entry.
level          : the level of compression of the new entry. (0 = Store method.)
comment        : add a comment for the file in the zip file header.
password       : set the password to protect this file.
useBz2         : use the bz2 compression algorithm. If false the zlib deflate algorithm will be
                 used. (Except MacOS/iOS/tvOS/watchOS/webGL)
ERROR CODES    : -1 = could not create or append
               : -2 = error during operation
               : -3 = failed to delete original entry
```

```
int replace_entry(string zipArchive, string arcFilename, byte[] newFileBuffer, int level = 9, string password = null, bool useBz2 = false);
```

A function that replaces an entry in a zip archive with a buffer. The original name of the archive will be used. Does not work on password protected archives.

```
zipArchive     : the full path to the zip archive
arcFilename    : the name of the file that will be replaced.
newFileBuffer  : a byte buffer that will replace the original entry.
level          : the level of compression of the new entry. (0 = Store method.)
password       : set the password to protect this file.
useBz2         : use the bz2 compression algorithm. If false the zlib deflate algorithm
                 will be used. (Except MacOS/iOS/tvOS/watchOS/webGL)
ERROR CODES    :  1 = success
               : -5 = failed to delete the original file
               : -6 = failed to append the buffer to the zip
```

```
int extract_entry(string zipArchive, string arcFilename, string outpath, object fileBuffer = null, ulong[] proc = null, string password = null);
```

A function that will extract only the specified file that resides in a zip archive.

```
zipArchive     : the full path to the zip archive from which we want to extract the specific file.
arcFilename    : the specific file we want to extract. (If the file resides in a directory, the
                 directory path should be included. like dir1/dir2/myfile.bin)
                 (on some zip files the internal dir structure uses \\ instead of / characters for
                 directories separators. In that case use the appropriate chars that will allow the
                 file to be extracted.)
outpath        : the full path to where the file should be extracted + the desired name for it.
fileBuffer     : A buffer that holds a zip file. When assigned the function will read from
                 this buffer and will ignore the filePath. Can be a c# byte[] buffer or an
                 unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the
                 zipArchive string parameter will be used to pass the Length of the Intptr
                 buffer!
proc           : a single item ulong array that gets updated with the progress of the decompression
                 in bytes. (100% is reached when the decompressed size of the file is reached.)
password       : if needed, the password to decrypt the entry.
ERROR CODES    : -1 = extraction failed
               : -2 = could not initialize zip archive.
               : -3 = could not locate entry
               : -4 = could not get entry info
               : -5 = password error
```

```
              : -6 = non valid zip size when using native file buffer
              : -7 = output directory does not exist or could not be created
              :  1 = success
```

```
int extract_entries(string zipArchive, string[] fileList, string outpath, object
fileBuffer = null, ulong[] proc = null, string password = null)
```

A function that will extract a list of entries from a zip file.

```
ZipArchive     : the full path to the zip archive from which we want to extract the specific files.
FileList[]     : a string array list of the entries we want to exctract.
Outpath        : the full path to where the entries should be extracted.
FileBuffer     : A buffer that holds a zip file. When assigned the function will read from this
                 buffer and will ignore the filePath.
               : Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an
                 IntPtr is used, the zipArchive string parameter will be used to pass the Length of
                 the Intptr buffer!
Proc           : a single item ulong array that gets updated with the progress of the decompression
                 in bytes. (100% is reached when the decompressed size of the file is reached.)
password       : if needed, the password to decrypt the entry.

ERROR CODES    : -2 = could not initialize zip archive.
               : -3 = fileList provided is null.
               : -4 = could not get entry info
                      : -5 = password error
               : -6 = non valid zip size when using native file buffer
               : -7 = output directory does not exist or could not be created
               :  1 = success
```

```
byte[][] entries2Buffers(string zipArchive, string[] entries, ref int progress, object
fileBuffer = null, string password = null)
```

A function that will decompress a list of entries to newly created byte buffers and returns them in the form of byte[][]

```
zipArchive       : the full path to the zip archive from which a specific file will be extracted to
                   a byte buffer.
entries          : the files we want to extract to the new buffers. (If the files resides in a
                   directory, the directory should be included.)
progress         : a referenced integer that will count how many entries have been extracted to the
                   buffers.
FileBuffer       : A buffer that holds a zip file. When assigned the function will read from this
                   buffer and will ignore the filePath.
                 : Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When
                   an IntPtr is used, then the zipArchive string parameter will be used to pass the
                   Length of the Intptr buffer!
password         : If the archive is encrypted use a password.

ERROR CODES      :  the byte[] buffers on success
                 :  null on failure
```

```
int decompress_File(string zipArchive, string outPath, int[] progress, object fileBuffer = null, ulong[] proc = null, string password = null);
```

A function that decompresses a zip file. If the zip contains directories, they will be created.

```
zipArchive     : the full path to the zip archive that will be decompressed.
outPath        : the directory in which the zip contents will be extracted.
progress       : a single item integer array that gets updated with the number of files decompressed.
                 To use it in realtime, call this function in a separate thread.
fileBuffer     : A buffer that holds a zip file. When assigned the function will read from
                 this buffer and will ignore the filePath. Can be a c# byte[] buffer or an
                 unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the
                 zipArchive string parameter will be used to pass the Length of the Intptr
                 buffer!
proc           : a single item ulong array that gets updated with the progress of the decompression
                 in bytes. (100% is reached when the decompressed size of the files is reached.)
password       : if needed, the password to decrypt the archive.
ERROR CODES    : -1 = could not initialize zip archive.
               : -2 = failed extraction
               : -6 = IntPtr buffer had no valid buffer length parameter passed.
               :  1 = success
```

```
int compress_File(int levelOfCompression, string zipArchive, string inFilePath, bool append = false, string fileName = "", string comment = null, string password = null, bool useBz2 = false, ulong[] byteProgress = null);
```

A function that compresses a file to a zip file. If the flag append is set to true then it will get appended to an existing zip file. This function is slow when appending many files. Use compress_File_List instead.

```
levelOfCompression    : (0-9) recommended 9 for maximum (0 = Store method.)
zipArchive            : the full path to the zip archive that will be created
inFilePath            : the full path to the file that should be compressed and added to the zip
                        file.
append                : set to true if you want the input file to get appended to an existing zip
                        file. (if the zip file does not exist it will be created.)
filename              : if you want the name of your file to be different then the one it has, set
                        it here. If you add a folder structure to it, like (dir1/dir2/myfile.bin)
                        the directories will be created in the zip file.
comment               : add a comment for the file in the zip file header.
password              : set the password to protect this file.
useBz2                : use the bz2 compression algorithm. If false the zlib deflate algorithm
                        will be used. (Except MacOS/iOS/tvOS/watchOS/webGL)
disksize              : if a disksize is used > 0 then the zip archive will be split to the
                        assigned disksize (in bytes).
byteProgress          : this variable is a single ulong array that keeps track of all the
                        uncompressed bytes that have been processed. (set to 0 after finish).
                        To get a progress of the compression, store the sum of the files that will
                        get zipped and compare it to byteProgress.
ERROR CODES           :  1 = success
                      : -1 = could not create or append
                      : -2 = error during operation
```

```
int compress_File_List(int levelOfCompression, string zipArchive, string[] inFilePath, int[] progress = null, bool append = false, string[] fileName = null, string password = null, bool useBz2 = false, ulong[] byteProgress = null);
```

*A function that compresses a list of files to a zip file. Use this function to compress multiple files **fast** instead of appending to existing files with the compress_File function.*

```
levelOfCompression    : (0-9) recommended 9 for maximum (0 = Store method.)
```

zipArchive : the full path to the zip archive that will be created
inFilePath[] : an array of the full paths to the files that should be compressed and added to the zip file.
progress : this var will increment until the number of the input files and this are equal.
append : set to true if you want the input file to get appended to an existing zip file. (if the zip file does not exist it will be created.)
filename[] : if you want the names of your files to be different then the one they have, set it here. If you add a folder structure to it, like (dir1/dir2/myfile.bin) the directories will be created in the zip file.
password : set the password to protect this file.
useBz2 : use the bz2 compression algorithm. If false the zlib deflate algorithm will be used. (Except MacOS/iOS/tvOS/watchOS/webGL)
disksize : if a disksize is used > 0 then the zip archive will be split to the assigned disksize (in bytes).
byteProgress : this variable is a single ulong array that keeps track of all the uncompressed bytes that have been processed. (set to 0 after finish). To get a progress of the compression, store the sum of the files that will get zipped and compare it to byteProgress.
ERROR CODES : 1 = success
: -1 = could not create or append
: -2 = error during operation

---

int compressDir(string sourceDir, int levelOfCompression, string zipArchive, bool includeRoot = false, string password = null, bool useBz2 = false, ulong[] byteProgress = null);

Compress a directory with all its files and subfolders to a zip file. (This function is way faster when adding manually multiple files to a zip with the compress_File function.)

sourceDir : the directory you want to compress
levelOfCompression : the level of compression (0-9). (0 = Store method.)
zipArchive : the full path+name to the zip file to be created .
includeRoot : set to true if you want the root folder of the directory to be included in the zip archive. Otherwise leave it to false.
password : set the password to protect this file.
useBz2 : use the bz2 compression algorithm. If false the zlib deflate algorithm will be used. (Except MacOS/iOS/tvOS/watchOS/webGL)
disksize : if a disksize is used > 0 then the zip archive will be split to the assigned disksize (in bytes).
byteProgress : this variable is a single ulong array that keeps track of all the uncompressed bytes that have been processed. (set to 0 after finish). To get a progress of the compression, store the sum of the files that will get zipped and compare it to byteProgress.
ERROR CODES : 1 = success
: -1 = could not create or append
: -2 = error during operation

If you want to get the progress of compression, call the getAllFiles function to get the total number of files in a directory and its subdirectories. The compressDir when called from a separate thread will update the public static int cProgress.
Divide this with the total number of files (as floats) and you have the % of the procedure.

The same can be done with the byteProgress variable. Store the sum of the files that will get zipped and compare it to byteProgress.

```
int getAllFiles(string Dir);
```

Use this utility function to get the total files of a directory and its subdirectories.

[Android, iOS, Linux, MacOSX only]

```
int setFilePermissions(string filePath, string _user, string _group, string _other);
```

Sets permissions of a file in user, group, other.
Each string should contain any or all chars of "rwx".
Returns 0 on success.

[Windows only]

```
bool setTarEncoding(uint encoding); // for tar archives (default UTF8)
```

*Set encoding of file names (read/write) on **Windows**.*

Usually the default setting (65001) is OK for most Unicode paths. If you are having issues with some special characters use the appropriate encoding.

CP_ACP = 0
CP_OEMCP/UNICODE = 1
CP_UTF8 = 65001 **(default)**
CP_WINUNICODE = 1200
Chinese = 54936 or 936

[Windows only]

```
void setEncoding(uint encoding); // for zip archives (default UTF8)
```

Set encoding of file names (read/write) on Windows.

Usually the default setting (65001) is OK for most Unicode paths. If you are having issues with some special characters use the appropriate encoding.

CP_ACP = 0
CP_OEMCP/UNICODE = 1
CP_UTF8 = 65001 **(default)**
CP_WINUNICODE = 1200
Chinese = 54936 or 936

```
bool validateFile(string zipArchive, object fileBuffer = null);
```

A function that will validate a zip archive.

zipArchive : the zip to be checked
fileBuffer : A buffer that holds a zip file. When assigned the function will read from this buffer and will ignore the filePath. Can be a c# byte[] buffer or an unmanaged Inptr buffer or an inMemory class! When an IntPtr is used, the zipArchive string parameter will be used to pass the Length of the Intptr buffer!
ERROR CODES : true. The archive is ok.
: false. The archive could not be validated.

InMemory SECTION

In order to use the inMemory zip file creation you must create a new **inMemory** class object like this: lzip.inMemory t = new lzip.inMemory(); (see demo script)

The class has the following public functions:

```
int size();
```

Use the size() function to get the size of the memory it occupies.

```
IntPtr memoryPointer();
```

Returns the native memory pointer of the class.

```
byte[] getZipBuffer();
```

The getZipBuffer() function returns a new byte[] buffer that contains the inmemory zip.

```
bool isClosed;
```

A bool that tells if an inMemory zip archive is open or closed. (Used with the low level functions only.)

```
void free_imemory(inMemory t);
```

Use this function to free the pointer and the object of the inMemory zip archive. It is important to call this function after you don't need the in memory zip any more!

Low level functions:
(Use these functions for faster inMemory zip file creation, especially when adding or appending multiple entries.)

```
bool inMemoryZipStart(inMemory t);
```

A function that creates an inMemory zip archive.

t : the inMemory class that holds the pointer to our inMemory zip file. You can call this function again for more buffers with the same inMemory object and the next buffers will get appended to the in memory zip.

Returns true on success.

```
int inMemoryZipAdd(inMemory t, int levelOfCompression, byte[] buffer, string fileName, string comment = null, string password = null, bool useBz2 = false);
```

A function that adds a buffer as a zip entry in an opened inMemory zip archive with the inMemoryZipStart function.

t	: the inMemory class that holds the pointer to our inMemory zip file.
levelOfCompression	: (0-9) recommended 9 for maximum (10 is highest but slower and not zlib compatible)
buffer	: The byte[] buffer that should be added to the zip.
filename	: The name of the file added. If you add a folder structure to it,like (dir1/dir2/myfile.bin) the directories will be created in the zip file.
comment	: add a comment for the file in the zip file header.
password	: set the password to protect this file.
useBz2	: use the bz2 compression algorithm. If false the zlib deflate algorithm will be used. (not available for MacOS/iOS/tvOS)

Returns 0 on success.

```
IntPtr inMemoryZipClose(inMemory t);
```

A function to close the inMemory zip archive that has been created with the inMemoryZipStart function.

t : the inMemory class that holds the pointer to our inMemory zip file.

If t.lastResult is different then 0 a null pointer will get returned.

Normal functions:

```
IntPtr compress_Buf2Mem(inMemory t, int levelOfCompression, byte[] buffer, string fileName = "", string comment = null, string password = null);
```

*A function that compresses a buffer to an inMemory zip file. Appending using this function can be slow. Use the low level functions for way **faster** processing.*

t	: the inMemory class that holds the pointer to our inMemory zip file. You can call this function again for more buffers with the same inMemory object and the next buffers will get appended to the in memory zip.
levelOfCompression	: (0-9) recommended 9 for maximum .
buffer	: The byte[] buffer that should be added to the zip.
filename	: The name of the file added. If you add a folder structure to it, like (dir1/dir2/myfile.bin) the directories will be created in the zip file.
comment	: add a comment for the file in the zip file header.
password	: set the password to protect this file.

Although the inMemory t.pointer gets internally updated, the function returns an IntPtr of the inMemory zip file buffer.
So to check if the operation was successful, check if the pointer returned is a non IntPtr.Zero pointer.

```
int decompress_Mem2File(inMemory t, string outPath, int[] progress, ulong[] proc = null,
string password = null);
```

A function that decompresses a zip file from an inMemory pointer. If the zip contains directories, they will be created.

```
t             : the inMemory class that holds the pointer to our inMemory zip file.
outPath       : the directory in which the zip contents will be extracted.
progress      : a single item integer array that increments with the archives that have been
                extracted. To use it in realtime, call this function in a separate thread.
proc          : a single item ulong array that gets updated with the progress of the decompression
                in bytes. (100% is reached when the compressed size of the file is reached.)
password      : if needed, the password to decrypt the archive.

ERROR CODES
              : -1 = could not initialize zip archive.
              : -2 = failed extraction
              :  1 = success
```

```
int entry2BufferMem(inMemory t, string entry, ref byte[] buffer, string password = null);
```

A function that will decompress a file from an inmemory zip file pointer directly in a provided byte buffer.

```
t             : the inMemory class that holds the pointer to our inMemory zip file.
entry         : the file we want to extract to a buffer. (If the file resides in a directory, the
                directory should be included.)
buffer        : a referenced byte buffer that will be resized and will be filled with the extraction
                data.
Password      : If the archive is encrypted use a password.

ERROR CODES   :  1 = success
              : -2 = could not find/open zip file
              : -3 = could not locate entry
              : -4 = could not get entry info
              : -5 = password error
              : -18 = the entry has no size
              : -104 = internal memory error
```

```
byte[] entry2BufferMem(inMemory t, string entry,  string password = null);
```

A function that will decompress a file from an inmemory zip file pointer to a new created and returned byte buffer.

```
t             : the inMemory class that holds the pointer to our inMemory zip file.
entry         : the file we want to extract to a buffer. (If the file resides in a directory, the
                directory should be included.
password      : If the archive is encrypted use a password.

ERROR CODES   : non-null  = success
              : null      = failed
```

```
int entry2FixedBufferMem(inMemory t, string entry, ref byte[] fixedBuffer, string password
= null);
```

A function that will decompress a file from an inmemory zip file pointer directly to a provided fixed size byte buffer.

Returns the uncompressed size of the entry.

```
t             : the inMemory class that holds the pointer to our inMemory zip file.
entry         : the file we want to extract to a buffer. (If the file resides in a directory, the
                directory should be included.)
buffer        : a referenced fixed size byte buffer that will be filled with the extraction data. It
                should be large enough to store the data.
password      : if the archive is encrypted use a password.

ERROR CODES   :  1 = success
              : -2 = could not find/open zip archive
              : -3 = could not locate entry
              : -4 = could not get entry info
              : -5 = password error
              : -18 = the entry has no size
              : -19 = the fixed size buffer is not big enough to store the uncompressed data
              : -104 = internal memory error
```

```
long getFileInfoMem(inMemory t);
```

This function fills the Lists with the filenames and file sizes that are in the inMemory zip file.

Returns the total size of uncompressed bytes of the files in the zip archive.

```
t             : the inMemory class that holds the pointer to our inMemory zip file.

ERROR CODES   : -1 = Input file not found
              : -2 = Could not get info
```

HIDDEN/MERGED ZIP ARCHIVE SECTION

The following functions allow some advanced operations on some special cases where the user has attached/merged/hidden a zip archive in another bigger file or memory buffer.

This is the extended zip entry struct that is used by the lzip.zinfo List to get extended info on zip archives, be it normal zips or hidden/merged.

```
public struct zipInfo{
    public short VersionMadeBy;
    public short MinimumVersionToExtract;
    public short BitFlag;
    public short CompressionMethod;
    public short FileLastModificationTime;
    public short FileLastModificationDate;
    public int CRC;
    public int CompressedSize;
    public int UncompressedSize;
    public short DiskNumberWhereFileStarts;
    public short InternalFileAttributes;
    public int ExternalFileAttributes;
    public int RelativeOffsetOfLocalFileHeader;
     public int AbsoluteOffsetOfLocalFileHeaderStore;
    public string filename;
    public string extraField;
    public string fileComment;
};
```

---

bool getZipInfo(string fileName);

This function is used to get extended info of the entries in a zip archive in the file system.
Use this as an alternative function to get zip info with more information about entries.
*For now it does not work with **zip64**.*
Returns true on success.

---

bool getZipInfoMerged(string fileName, ref int pos, ref int size, bool getCentralDirectory = false);

Get position, size and/or entry info of a zip archive in the file system.
This function is mainly useful to discover a zip archive hidden or merged in another bigger archive.
filename: the path to the archive.
ref pos: the position in bytes of the zip archive.
ref size: the size of the zip archive.
If **getCentralDirectory** is set to true it will fill the zinfo List with extended entry information.
*For now it does not work with **zip64**.*
Returns true on success.

```
bool getZipInfoMerged(byte[] buffer, ref int pos, ref int size, bool getCentralDirectory =
false);
```

Get position, size and/or entry info of a zip archive in a buffer.
This function is mainly useful to discover a zip archive hidden or merged in another bigger buffer.
buffer: the buffer where the merged zip archive resides.
ref pos: the position in bytes of the zip archive.
ref size: the size of the zip archive.
If **getCentralDirectory** is set to true it will fill the zinfo List with extended entry information.
*For now it does not work with **zip64**.*
Returns true on success.

```
bool getZipInfoMerged(byte[] buffer);
```

This function is used to get extended info of the entries in a zip archive in a buffer.
buffer: the buffer where the merged zip archive resides.
*For now it does not work with **zip64**.*
Returns true on success.

```
byte[] getMergedZip(string filePath, ref int position, ref int siz);
```

Get the merged zip archive in a file system archive as a byte buffer and return position and size.
filePath: the path to the archive.
ref position: the position in bytes of the zip archive.
ref size: the size of the zip archive.

```
byte[] getMergedZip(string filePath);
```

Get the merged zip archive in a file system archive as a byte buffer.
filePath: the path to the archive.

```
byte[] getMergedZip(byte[] buffer, ref int position, ref int siz);
```

Get the merged zip archive in a buffer as a byte buffer with size and position info.
buffer: the buffer where the zip archive resides.
ref position: the position in bytes of the zip archive.
ref size: the size of the zip archive.

```
byte[] getMergedZip(byte[] buffer);
```

Get the merged zip archive in a buffer as a byte buffer.
buffer: the buffer where the zip archive resides.

```
int decompressZipMerged(string file, string outPath, int[] progress, ulong[] proc = null,
string password = null);
```

A function that extracts all contents of a zip file that is merged in another file in the file system, to disk.

file : the path to the file where the zip archive resides.
outPath : the directory in which the zip contents will be extracted.
progress : a single item integer array that increments with the archives that have been extracted. To use it in realtime, call this function in a separate thread.
proc: : a single item ulong array that gets updated with the progress of the decompression in bytes. (100% is reached when the compressed size of the file is reached.)
password : if needed, the password to decrypt the archive.
ERROR CODES
: -1 = could not initialize zip archive.
: -2 = failed extraction
: 1 = success

```
int decompressZipMerged(byte[] buffer, string outPath, int[] progress, ulong[] proc =
null, string password = null);
```

A function that extracts all contents of a merged zip file that resides in a buffer to disk.

buffer : the buffer where the zip archive resides.
outPath : the directory in which the zip contents will be extracted.
progress : provide a single item integer array to write the current index of the file getting extracted. To use it in realtime, call
: this function in a separate thread.
proc: : a single item ulong array that gets updated with the progress of the decompression in bytes. (100% is reached when the compressed size of the file is reached.)
password : if needed, the password to decrypt the archive.
ERROR CODES
: -1 = could not initialize zip archive.
: -2 = failed extraction
: *1 = success*

```
int entry2FileMerged(string file, string entry, string outPath, string overrideEntryName =
null, string password = null);
```

Extract an entry from a merged zip that resides in the file system to disk.

Returns 1 on success.
file : the path to the file where the zip archive resides.
entry : the entry to extract.
outPath : the path where the entry will be extracted.
overrideEntryName : with this you can override the default entry name.
password : if needed, the password to decrypt the archive.

```
int entry2FileMerged(byte[] buffer, string entry, string outPath, string overrideEntryName
= null, string password = null);
```

Extract an entry from a merged zip that resides in a buffer to disk.

Returns 1 on success.

buffer	: the buffer where the zip archive resides.
entry	: the entry to extract.
outPath	: the path where the entry will be extracted.
overrideEntryName	: with this you can override the default entry name.
password	*: if needed, the password to decrypt the archive.*

```
byte[] entry2BufferMerged(byte[] buffer, string entry, string password = null);
```

A function that extracts an entry from a zip archive that is merged/hidden in a buffer and returns the extracted data in a new buffer.

buffer	: the buffer where the zip archive resides.
entry	: the entry to extract.
password	: if needed, the password to decrypt the archive.

```
int entry2BufferMerged(byte[] buffer, string entry, ref byte[] refBuffer, string password
= null);
```

A function that extracts an entry from a zip archive that is merged/hidden in a buffer and returns the extracted data in a referenced buffer that will get resized to fit the data.

buffer	: the buffer where the zip archive resides.
refBuffer	: the referenced buffer that will get resized to store the decompressed data.
entry	: the entry to extract.
password	: if needed, the password to decrypt the archive.

```
int entry2FixedBufferMerged(byte[] buffer, string entry, ref byte[] fixedBuffer, string
password = null);
```

A function that extracts an entry from a zip archive that is merged/hidden in a buffer and writes the extracted data in a fixed size buffer.

Returns the size of the uncompressed data.

buffer	: the buffer where the zip archive resides.
entry	: the entry to extract.
fixedBuffer	: the fixed sized buffer where the data will be written.
password	: if needed, the password to decrypt the archive.
ERROR CODES	
	: -1 = could not initialize zip archive.
	: -2 = failed extraction
	: 1 = success

```
byte[] entry2BufferMerged(string file, string entry, string password = null);
```

A function that extracts an entry from a zip archive that is merged/hidden in the file system and returns the extracted data in a new buffer.

file : the path to the file where the zip archive resides.
entry : the entry to extract.
password : if needed, the password to decrypt the archive.

```
int entry2BufferMerged(string file, string entry, ref byte[] refBuffer, string password = null);
```

A function that extracts an entry from a zip archive that is merged/hidden in the file system and returns the extracted data in a referenced buffer that will get resized to fit the data.

file : the path to the file where the zip archive resides.
entry : the entry to extract.
***refB*uffer** : the referenced buffer that will get resized to store the decompressed data.
password : if needed, the password to decrypt the archive.

```
int entry2FixedBufferMerged(string file, string entry, ref byte[] fixedBuffer, string password = null);
```

A function that extracts an entry from a zip archive that is merged/hidden in the file system and writes the extracted data in a fixed size buffer.

file : the path to the file where the zip archive resides.
entry : the entry to extract.
fixedBuffer : the fixed sized buffer where the data will be written.
password : if needed, the password to decrypt the archive.

GZIP SECTION

```
int gzip(byte[] source, byte[] outBuffer, int level, bool addHeader = true, bool addFooter = true, bool overrideDateTimeWithLength = false);
```

Compress a byte buffer to gzip format.
Returns the size of the compressed buffer.

source : the uncompressed input buffer.
outBuffer : the provided output buffer where the compressed data will be stored (it should be at least the size of the input buffer +18 bytes).
level : the level of compression (0-9)
addHeader : if a gzip header should be added. (recommended if you want to write out a gzip file)
addFooter : if a gzip footer should be added. (recommended if you want to write out a gzip file)

overrideDateTimeWithLength: use the bytes 5-8 of the header to store the gzip length instead of DateTime modification. This is useful when you want to know the compressed size of a gzip buffer. Then use the gzipCompressedSize function to get this size.

If you add a header and a footer to the buffer, you can write out a file that is a regular gzip file.

```
int gzipUncompressedSize(byte[] source);
```

Get the uncompressed size from a gzip buffer that has a footer included.

source: the gzip compressed input buffer. (it should have at least a gzip footer).

```
int gzipCompressedSize(byte[] source, int offset = 0);
```

Get the compressed size of a gzip, if the compressed size of it has been written in the date header bytes and marked as such, with the gzip function above.

source: the gzip compressed input buffer.

```
int unGzip(byte[] source, byte[] outBuffer, bool hasHeader = true, bool hasFooter = true);
```

Decompress a gzip buffer.

returns : uncompressed size. negative error code on error.
source : the gzip compressed input buffer.
outBuffer : the provided output buffer where the uncompressed data will be stored.
hasHeader : if the buffer has a header.
hasFooter : if the buffer has a footer.

```
int unGzip2(object source, byte[] outBuffer, int intPtrLength = 0);
```

Decompress a gzip buffer. (This function assumes that the gzip buffer has a gzip header !!!)

returns : uncompressed size. negative error code on error. -11 when a native pointer is passed as source and no source length defined.
source : the gzip compressed input buffer. Can be byte[] or IntPtr pointer.
outBuffer : the provided output buffer where the uncompressed data will be stored.
IntPtrLength : when passing an IntPtr pointer as the input gzip, we need to pass the size of this buffer to the function. Not needed for byte[] source gz.

Merged gzip functions

```
int findGzStart(byte[] buffer);
```

Find where the merged gzip starts in a buffer.

buffer: a memory buffer that has a gzip merged at the end of it.

```
unGzip2Merged(byte[] source, int offset, int bufferLength, byte[] outBuffer);
```

Decompress a gzip buffer that is merged in the end of a buffer (This function assumes that the gzip buffer has a gzip header !!!)

returns: uncompressed size. negative error code on error.
source: the buffer where the gzip compressed input buffer resides. (at the end of it, or anywhere if you know the length of it)
offset: the offset in bytes where the gzip starts.
bufferLength: the length of the gzip buffer.
outBuffer: the provided output buffer where the uncompressed data will be stored.

Gzip file functions

```
int gzipFile(string inFile, string outFile, int level, ulong[] progress = null, bool
addHeader = true);
```

Create a gzip file in the file system.

returns 1 on success.

```
inFile:      the input file to compress.
outFile:     the output path of the created gzip.
level:       level of compression (0 - 10).
progress:    a ulong single item array that will report how many bytes have been processed. It
             should equal the uncompressed size.
addHeader:   if gzip header and footer should be added. If set to false the gzip can still be
             extracted but not opened from other decompression apps.

error codes  : -10 could not open input file
             : -11 could not write output file
             :  -1 general error
             :  -8 canceled.
```

```
int ungzipFile(string inFile, string outFile, ulong[] progress = null);
```

Decompress a gzip file.

returns 1 on success.

```
inFile:       the gzip file to extract.
outFile:      the output path of the extracted file.
progress:     a ulong single item array that will report how many bytes have been processed. It
              should equal the compressed size.
error codes  : -11 could not open input file
             : -12 could not write output file
             :  -3 error reading gz file
             :  -4 error writing output
             :  -8 canceled.
long ungzipBuffer2File(byte[] fileBuffer, string outFile, ulong[] progress = null)
```

Decompress a gzip file from a byte[] buffer to disk

returns number of decompressed bytes on success. -8 when canceled.

```
fileBuffer   : the gzip that is loaded into a byte[] buffer.
outFile      : the output path of the extracted file.
progress     : a ulong single item array that will report how many bytes have been processed. It
               should equal the compressed size of the gz file.

error codes  : -11 could not open input file
             : -12 could not write output file
             :  -3 error reading gz file
             :  -4 error writing output
             :  -8 canceled. (Make sure you delete the canceled output file)
```

TAR SECTION:

```
int tarExtract(string inFile, string outPath = null, int[] progress = null, ulong[]
byteProgress = null);
```

Untar a .tar archive

```
inFile       : the full path to the tar archive.
outPath      : the path where the extraction will take place. If null, the same path as the one of
               the inFile will be used.
progress     : a single item integer array that will get updated with the number of the entries that
               get extracted. Use in a Thread for real time report.
byteProgress : a single item ulong array that will get updated with the bytes that have been
               extracted so far. Use in a Thread for real time report.

Error codes   : -1 could not find input file.
              : -3 could not write output file.
              : -8 canceled
              :  1 Success
```

```
int tarExtractEntry(string inFile, string entry, string outPath = null, bool fullPaths =
true, ulong[] byteProgress = null);
```

Extract an entry from a tar archive

```
inFile       : the full path to our tar archive
entry        : the entry we want to extract. (If the file resides in a directory, the directory path
               should be included. like dir1/dir2/myfile.bin)
outPath      : the path in which want to extract our entry. If null the same path as the inFile will
               be used.
fullPaths    : if the entry resides in a directory, use this flag to create the directory structure
               or not. ! If it is set to false, you can use an absolute path in the outPath
               parameter to extract with a different filename !
byteProgress : a sigle item ulong array that will get updated with the bytes of the extraction
               progress. Use in a Thread for real time report.

Error codes   : -1 could not find input file.
              : -3 could not write output file.
              : -5 could not find entry
              : -8 canceled
              :  1 Success
```

```
int tarDir(string sourceDir, string outFile = null, bool includeRoot = false, int[]
progress = null, ulong[] byteProgress = null);
```

Create a tar file out of a directory containing files.

```
sourceDir        : the directory that contains our files
outFile          : the full path to the tar archive that will be created. If null the same name as
                   the sourceDir will be used.
includeRoot      : if the root directory should be included in the filenames.
progress         : a single item integer array that will increment with each file added to the tar
                   archive. Use in a Thread for real time report.
byteProgress     : a sigle item ulong array that will get updated with the bytes that get added to
                   the tar archive. Use in a Thread for real time report.

Error codes  : -1 could not write output file.
             : -3 could not find input file.
             : -8 canceled
             :  0 no files found in Dir.
```

```
int tarList(string outFile, string[] inFilePath,  string[] fileName = null, int[] progress
= null, ulong[] byteProgress = null);
```

This function creates a tar archive from a list of file paths provided.

```
outFile           : the full path to the tar archive that will be created.
inFilePath[]      : an array of the full paths to the files that should be added to the tar archive.
fileName[]        : if you want the names of your files to be different then the one they have, set
                    it here.
progress          : this var will increment until the number of the input files and this are equal.
                    Use in a Thread for real time report.
byteProgress      : this variable is a single ulong array that keeps track of all the bytes that
                    have been processed. Use in a Thread for real time report.

Error codes   : -1 could not write output file.
              : -3 could not find input file.
              : -4 input files number different than filenames number
              : -8 canceled
              : -10 an input file was not found.
              :  1 Success
```

```
UInt64 getTarInfo(string tarArchive);
```

This function fills the same Lists as the getFileInfo for zip, with the filenames and file sizes that are in the tar file. ***(See page 12)***

```
Returns       : the total size of uncompressed bytes of the files in the tar archive
tarArchive    : the full path to the archive, including the archives name. (/myPath/myArchive.tar)
Error codes   : 0 = Input file not found or  Could not get info.
```

*On windows (only) you can use the setTarEncoding function to set the character encoding of the file names.

BZIP2 SECTION:

```
int bz2Create(string inFile, string outFile = null, int level = 9, ulong[] byteProgress =
null);
```

Create a bz2 file in the file system.

returns 1 on success.

```
inFile:      the input file to compress.
outFile:     the output path of the created bz2. If null, the input file + ".bz2" will be used.
level:       level of compression (0 - 9).
progress:    a ulong single item array that will report how many bytes have been processed. It
             should equal the uncompressed size.

error codes   :  1 success
              : -3 could not read input file
              : -4 could not create output file
              : -8 canceled
```

```
int bz2Decompress(string inFile, string outFile = null, ulong[] byteProgress = null);
```

Decompress a bz2 file

returns 1 on success.

```
inFile       : the bz2 file to extract.
outFile      : the output path of the extracted file. If null, the original name will be used.
progress     : a ulong single item array that will report how many bytes have been processed. It
               should equal the compressed size of the bz2 file.

error codes  :  1 success
             : -1 could not create output file
             : -2 could not read input file
             : -8 canceled
```

Helper function

```
IEnumerator downloadZipFileNative(string url, Action<bool> downloadDone, Action<lzip.inMemory>
inmem, Action<IntPtr> pointer = null, Action<int> fileSize = null);
```

A Coroutine to download a file to a native/unmaged memory buffer.
You can call it for an lzip.inMemory class or for an IntPtr.
See testZip.cs or testWebGLtvOS.cs for usage examples.

*This function can only be called for **one file at a time. Don't use it to call multiple files at once.***

*This is useful to **avoid memory spikes** when downloading large files and intend to decompress from memory.*
With the old method, a copy of the downloaded file to memory would be produced by pinning the buffer to memory.
Now with this method, it is downloaded to memory and an inMemory zip structure is created. You can decompress from there, with no memory spikes and delete the inMemory zip, or keep it for further usage.

In any case, if you don't need the created inMemory zip, you should use the lzip.free_inmemory or lzip.releaseBuffer functions to free the memory!

You can modify this function or replace it with one that suits you better.
Keep in mind that the server where your files are should allow reporting of file sizes as defined in the .htaccess server file.

Parameters:

url:	The url of the file you want to download to a native memory buffer.
downloadDone:	Informs a bool that the download of the file to memory is done.
inmem:	An lzip.inMemory class to get the data.
pointer:	An IntPtr for a native memory buffer
fileSize:	The size of the downloaded file will be returned here.

SUPPORT:

For any questions, problems and suggestions please use this email address: elias_t@yahoo.com
Forum: http://forum.unity3d.com/threads/released-zip-native-multiplatfrom-plugin.339482