Thank you for purchasing this shared library for Android, iOS, Windows, OSX, Linux & webGL*.

The scope of this library is to have fast 7z decompression on Android, iOS, Windows, OSX, Linux.

The examples require some files that reside in the StreamingAssets folder. They are there just for testing purposes. In your final projects make sure you delete them so that you don't increase your build size.

The ios libraries are compiled as universal and bitcode enabled. That means that they will support 32 and 64 bit builds.
(**non-bitcode enabled iOS plugins are provided. If you are creating a non bitcode enabled project please use these provided plugins!**)
OSX bundle is compiled now as 64-bit only, since Apple store requires it.
The Windows and Linux libraries are compiled for x86 and x86_64 build modes.
The Android lib is compiled for armeabi-v7a, x86 and arm64-v8a.

***webGL support is for compressing/decompressing lzma buffers only.**

FEATURES:

- The library does 7z decompression and not 7z compression.
Compression of lzma alone files is supported.
Passwords are not supported.

- It is about 2.5x times faster then using a c# implementation for 7z decompression.

- You can extract a single file out of the 7z archive.

- If you intend to decompress large files it would be better to use the largeFiles flag.(consumes less ram)

- You can extract the contents of the 7z file keeping its folder structure.

- Ability to get the filenames and file sizes of files in a 7z archive.

- Get progress of extraction when the 7zip archive has multiple files.

- Ability to encode/decode to/from .lzma alone format.

- Ability to decode a specific file in a 7z archive to a byte buffer.

- Ability to decode/encode a byte buffer to/from the lzma alone format. (This function is very useful if you have stored some files in the lzma format on the web or in the Resources folder and you want to decompress them
in a buffer and not to save them in the Application.persistentDataPath folder.)

- Linux, iOS, Android, MacOSX can treat buffers 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.

** When the total uncompressed size of the files in a 7z archive are >2GB it is recommended to compress the 7z as non-solid. Otherwise the plugin could crash. **

!!! If you want to use only the 7zip plugin, please delete all the other plugin files or use the single packages in from the _plugin_packages folder!!!

>>> *** Important info *** <<<

In general the plugin will perform certain operations faster (like extract an entry, get info) when a 7z archive that contains multiple files, is compressed with '**Solid-Block size: non-solid**'.

If you have a big 7z file and want to extract only some files from it is Recommended to have it comrpessed as **non-solid**!

However the non-solid compression on multiple files will produce larger files.

A known issue is that when decompressing a 7z archive with multiple files and solid compression, the progress of the decompression gets delayed.

If you want to have more control over the decompression progress and keep your 7z files small, I recommend to compress your files in 4-5 7z archives with solid compression. This way you will have a rough control over progress and high compression values.

INSTRUCTIONS:

If you want to run a small example, compile and run the testScene.
It will download a small 7z file and it will decompress 2 small text files in your Application.persistentDataPath directory. You can easily set it up to download and decompress one file of your own server.

Additionally it will perform all the other functions that the plugin supports such as lzma encoding/decoding, getting 7z file content info, decode to buffer and decode/encode a byte buffer to/from the lzma alone format.

If you want to handle a file from your Resources or Streaming Assets folder on Android, copy it first to the Application.persistentDataPath folder and manipulate it there.
(Demo code included in the test Scene.)

FUNCTIONS:

int doDecompress7zip(string filePath, string exctractionPath, int[] progress, bool largeFiles=false, bool fullPaths=false, string entry=null, byte[] FileBuffer = null);

important: if you extract the same file again in the same path, remember to delete the previous extracted files first!
(athough the plugin supports overwrite, on some platforms it might not work)

filePath : the full path to the archive, including the archives name.(/myPath/myArchive.7z)
exctractionPath: the path in where you want your files to be extracted
progress : a single item integer array to get the progress of the extracted files (use this function when calling from a separate thread, otherwise call the 2nd implementation) (for ios this integer is not properly updated. So we use the lzma.getProgressCount() function to get the progress. See example.)
largeFiles : set this to true if you are extracting files larger then 50-60 Mb. It is slower though but prevents crashing your app when extracting large files!
fullPaths : set this to true if you want to keep the folder structure of the 7z file.
entry : set the name of a single file you want to extract from your archive. If the file resides in a folder, the full path should be added. (for example game/meshes/small/table.mesh). Files compressed with solid compression have slow extraction times of entries.
FileBuffer : A buffer that holds a 7zip file. When assigned the function will decompress from this buffer and will ignore the filePath. (Linux, iOS, Android, MacOSX)

Use this function from a separate thread to get the progress of the extracted files in the provided 'progress' integer array.

---

int doDecompress7zip(string filePath, string exctractionPath, bool largeFiles = false, bool fullPaths = false, string entry = null, byte[] FileBuffer = null);

Same as above only the progress integer array is a local variable.

Use this when you don't want to get the progress of the extracted files and when not calling the function from a separate thread.
Any existing files with the same file name will be overwritten.

See lzma.cs in the Plugins folder for error codes that are returned!
The SevenZipTest.cs file contains more useful comments.

---

public static string persitentDataPath="";

*If you want to be able to call the functions: **get7zinfo, get7zSize, decode2Buffer** from a **thread** set this string before to the Application.persistentDataPath !*

---

void setProps(int level = 5, int dictSize = 16777216, int lc = 3, int lp = 0, int pb = 2, int fb = 32, int numThreads = 2);

A function that sets the compression properties for the lzma compressor. Will affect the lzma alone file and the lzma buffer compression.
A simple usage of this function is to call it only with the 1st parameter that sets the compression level: lzma.setProps(9);
Lower dictionary compresses faster and consumes less ram!
!!! Multithread safe advice: call this function before starting any thread operations !!!

---

long get7zInfo(string filePath, string tempPath = null, byte[] FileBuffer = null);

This function reads the contents of a 7z archive and fills 2 ArrayLists with the file names and the files sizes of the included files.
Afterwards read the arraylists to handle this info. (see example)
It returns the uncompressed size in bytes of all the included files in the 7z archive.

filePath : the full path to the archive, including the archives name.
tempPath : a temp path that will be used to write the files info
FileBuffer : A buffer that holds a 7zip file. When assigned the function will read from this buffer and will ignore the filePath. (Linux, iOS, Android, MacOSX)

---

long get7zSize(string filePath=null, string fileName=null, string tempPath=null, byte[] FileBuffer=null);

This function returns the uncompressed file size of a given file in the 7z archive if specified, otherwise it will return the total uncompressed size of all the files in the archive. (see example)

filePath : the full path to the archive, including the archives name. (/myPath/myArchive.7z) if you call the function with filePath as null, it will try to find file sizes from the last call.
tempPath : a temp path that will be used to write the files info
fileName : the file name we want to get the file size (if it resides in a folder add the folder path also)
FileBuffer : A buffer that holds a 7zip file. When assigned the function will read from this buffer and will ignore the filePath. (Linux, iOS, Android, MacOSX)

*Files compressed with **solid** compression have slow access times.*

---

int LzmaUtilEncode(string inPath, string outPath);

This function encodes a single archive in lzma alone format.

inPath : the file to be encoded. (use full path + file name)
outPath : the .lzma file that will be produced. (use full path + file name)

*You can set the compression properties by calling the **setProps** function before.*
setProps(9) for example will set compression evel to highest level.

---

int LzmaUtilDecode(string inPath, string outPath);

This function decodes a single archive in lzma alone format.

inPath : the .lzma file that will be decoded. (use full path + file name)
outPath: the decoded file. (use full path + file name)

---

byte[] decode2Buffer(string filePath, string entry, string tempPath=null, byte[] FileBuffer=null);

This function decodes a specific archive in a 7z archive to a byte buffer.

filePath : the full path to the 7z archive.
entry : the file name to decode to a buffer. If the file resides in a folder, the full path should be added. Files compressed with solid compression have slow extraction times of entries.
tempPath : (optional) a temp path that will be used to write the files info (otherwise the path of the 7z archive will be used)
FileBuffer : A buffer that holds a 7zip file. When assigned the function will read from this buffer and will ignore the filePath. (Linux, iOS, Android, MacOSX)

---

bool compressBuffer(byte[] inBuffer, ref byte[] outBuffer, bool makeHeader=true);

This function encodes inBuffer to lzma alone format into the outBuffer provided.
The buffer can be saved also into a file and can be opened by applications that open the lzma alone format.
This buffer can be uncompressed by the decompressBuffer function.
Returns true if success
if makeHeader == false then the lzma 13 bytes header will not be added to the buffer. However if you want to decompress this buffer without the header included you have to feed the decompressBuffer function with the known uncompressed size.

---

bool compressBufferPartial(byte[] inBuffer, int inBufferPartialIndex, int inBufferPartialLength, ref byte[] outBuffer, bool makeHeader = true);

Same as the above function, only it compresses a part of the input buffer.

inBufferPartialLength : the size of the input buffer that should be compressed
inBufferPartialIndex : the offset of the input buffer from where the compression will start

---

int compressBufferPartialFixed(byte[] inBuffer, int inBufferPartialIndex, int inBufferPartialLength, ref byte[] outBuffer, bool safe = true, bool makeHeader = true);

*Same as **compressBufferPartial**, only this function will compress the data into a fixed size buffer.*
The compressed size is returned so you can manipulate it at will.

---

int compressBufferFixed(byte[] inBuffer, ref byte[] outBuffer, bool safe = true, bool makeHeader=true);

*Same as the **compressBuffer** function, only this one 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.

Use the safe flag if you want to abort the operation if the result is larger then the fixed buffer.
Otherwise the fixed buffer will be filled partially.

---

int decompressAssetBundle(byte[] inBuffer, ref byte[] outbuffer);

This function will decompress a compressed asset bundle.
It finds the magic number of the lzma format and extracts from there.
(works only on the old lzma format.)

---

int decompressBuffer(byte[] inBuffer, ref byte[] outbuffer, bool useHeader=true, int customLength=0);

This function decompresses an lzma compressed byte buffer.
If the useHeader flag is false you have to provide the uncompressed size of the buffer via the customLength integer.

---

int decompressBufferFixed(byte[] inBuffer, ref byte[] outbuffer, bool safe = true, bool useHeader=true, int customLength=0);

Same as above 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.

Use the safe flag if you want to abort the operation if the result is larger then the fixed buffer.
Otherwise the fixed buffer will be filled partially.

[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.

SUPPORT:

For any questions, problems and suggestions please use this email address: elias_t@yahoo.com

forum: http://forum.unity3d.com/threads/7zip-lzma-and-zip-native-multiplatform-plugins.211273/