Understanding 7z Compression File Format
written by gordon@romvault.com

To understand the data structures used in the 7z file format you must first understand how 7z internally works.

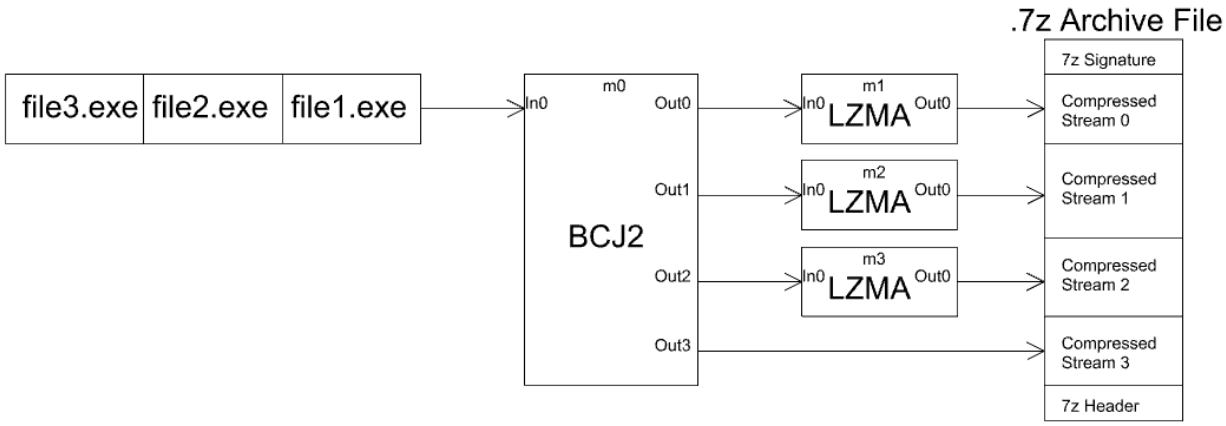
7z has 2 concepts that make it different from many other compression systems:
The first is that 7z can take many input files and concatenate them together and then treat them as a single stream of data. This can lead to a much higher compression ratio compared to compressing each file individually.

The second concept is that 7z can chain multiple compression algorithms and pre-processing filters together to gain even higher compression ratios.

Here is an example of these two concepts working together:

7z a -t7z archive.7z *.exe -m0=BCJ2 -m1=LZMA:d23 -m2=LZMA:d19 -m3=LZMA:d19 -mb0s0:1 -mb0s1:2 -mb0s2:3

This example is using the 7z.exe command line to make a 7z archive called archive.7z and is adding a number of exe files into the archive.
First of all the .exe's files in the current directory will be internally concatenated together by 7z into a single input stream.



This sample then contains 4 compression routines listed as m0 to m3.
m0 is a BCJ2 compression routine. (The exact details of this routine are not important here, but more on that in a bit.) m1, m2 & m3 are 3 independent LZMA compression routines.

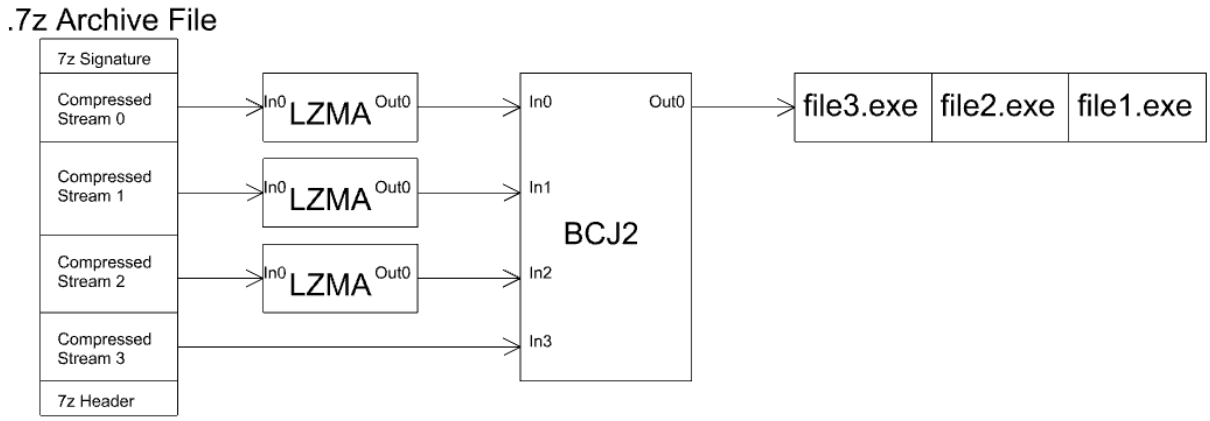
What you do need to know is that BCJ2 compression takes one input stream (in this case all the .exe files concatenated together) and actually produces 4 output streams.

So what we actually want to do is take the first three output streams and put each one into its own LZMA compression routine, and also take the fourth output stream and just directly store it without any further compression.

The compression routines are chained together in the command line example with the –mb options:
-mb0s0:1 says take m0 (BCJ2) output 0 and feed it into m1 (LZMA) input 0
-mb0s1:2 says take m0 (BCJ2) output 1 and feed it into m2 (LZMA) input 0
-mb0s2:3 says take m0 (BCJ2) output 2 and feed it into m3 (LZMA) input 0
And in this example m0 (BCJ2) output 3 is not linked to anything so it just streams directly out.

So in this example regardless of how many exe files are being compressed we will store 4 streams of data inside the 7z file.

Now to uncompress these files we have the reverse setup.



To decompress this 7z file the 4 streams will be reopened. The first streams 3 will be fed into separate LZMA decompression routines, and then the output of these 3 LZMA's and the fourth stream direct from the 7z file will all be fed back into the BCJ2 decompression routine which will then output a single data stream.

7z will have also stored the name, size and order of the original .exe files that are stored in this data stream so it can then split this stream back up and output the original decompressed .exe file.

One more detail should be added here: If a 7z file exists and you want to add more files to it, most times you do not extend the existing streams as this would require uncompressing all of the data in those streams and then re-compressing everything including the new file. The common thing to do is to create new stream(s) for the new files, which would also require a new copy of the uncompress routines required to process these new files (which could be very different from the files already in the 7z file.) Internally in 7z this is where the 'Folders' storage structure is used, as each time you add a new set of files a new Folder is made to store all the required stream data about those files.

7z Indroduction to the Data Structure

So now please look ahead at the 7z Data Structure a few page down, and get a basic feel for its structure. In this section I will explain how the previous example is stored in the data Structured, and as part of this we will define many of the terms used in the data structure.

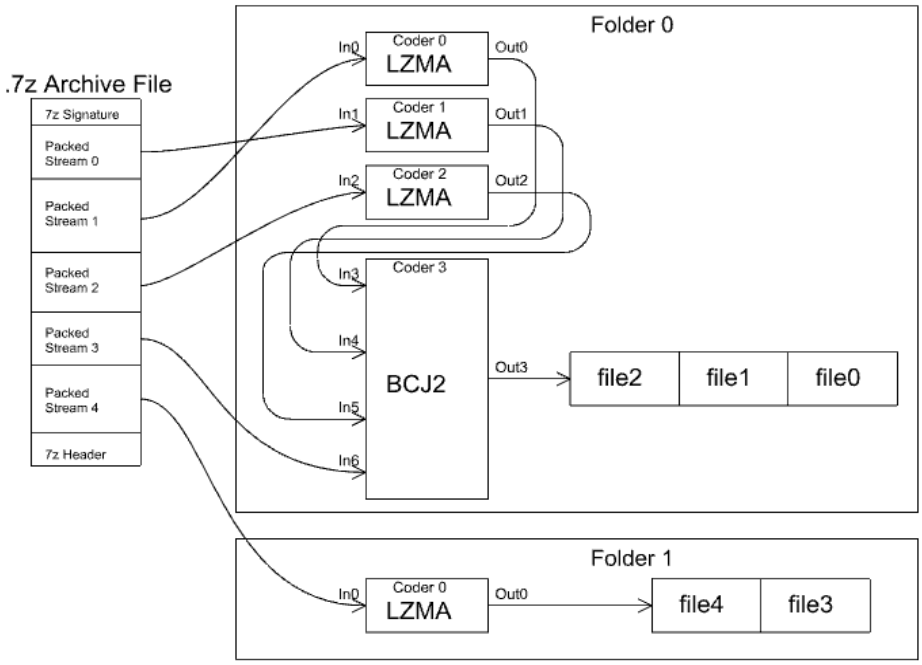
First up is the **Header** this contains just two variables: **FileInfo** and **StreamsInfo.**

FileInfo is probably the easiest structure to understand. It contains the name of each file in the 7z and a couple of other flags defining if the file is actually a directory or if the file is an empty file. The directories and empty files are special cases as they need to exist in the 7z but are not associated with any uncompression streams. (need to define the flags a little more clearly here.)

StreamsInfo this contains three variables:
The first **PackPosition** is simple a relative file pointer to the first stream of data in the 7z file (relative to the end of the 7z signature header.)
The second is an array of **PackedStreamsInfo**'s a PackedStream is what I referred to above as a Compressed Stream. So this is an array of every PackedStream in the 7z file. (Of which there are 4 in the above example.)
The third is an array of **Folders.** The **folders** is where all the data about the compression systems being used is stored and is a very complex set of data. There is an array of **Folders** as explained at the end of the example above where files have been added to the 7z file at multiple different times, in which case each file set being added would gets its own folder about the compression system used for that set of files and its associated streams. Note that the PackedStreamsInfo is the complete list of PackedStreams in the 7z file, these PackedStreams may be used across different Folders in the Folders array.

PackedStreamsInfo contains location information about a packedStream, there are three varaibles:
PackedSize simple the length of the packedStream.
CRC the crc of the packedStream. This is optional and normally not stored.
StreamPosition this is the file pointer to the start of this stream relative to **StreamsInfo.PackPosition**

So that is page 1 of **7z Data Structure** covered, that was the easy part. To understand Folders it is best we go back to our first example, and look at what data needs to be stored to represent this uncompression system, and then see how this data fits into the actual 7z folder/coder/bindpair structure.



This diagram shows the results of the full example above, where we first added 3 files using the example command line, and then later we added 2 more files just using LZMA compression.

So in this example **SteamsInfo** we have 2 **Folders** in the **Folders** array. So for now we will just look at Folder 0. Folder 0 contains 4 Coders, the 3 LZMA coders and the 1 BCJ2 coder.

The first variable stored in a **folder** is an array of **Coder**. Storing the details of these 4 coders. We will fully describe the Coders later.

The **BindingPairs** array is next. This stores the information needed to link the outputs of the coders to the inputs of other coders. So a **bindingpair** is just an output index and an input index. You will see in the above diagram that we are now numbering the inputs and outputs of the codes at a global level counting all of the inputs and outputs of the coders inside of this folder. So in this example:
Output 0 is linked to Input 3, Output 1 is linked to Input 4 and Output 2 is linked to Input 5.

Next is the **PackedStreamIndicies** array. This connects the Packed Streams to the remaining inputs of the codes. Remaining inputs means coders inputs that are not already linked by the BindingPairs. So the remaining inputs in this example are In0, In1, In2 & In6. So there will be 4 entries in the PackedStreamIndicies array, linking the PackedStream numbers to the remaining Inputs in order. So in this example the array will be 1,0,2,3.

UnpackedStreamSizes this examples has 4 Output Streams (Out0 through Out3) this array stores the expected output size of these 4 unpacked streams.

The goal of any folder is to have one final output stream, in this case Out3. All the other Out's should be linked back in somewhere with the BindingPairs. The remaining one output is the final output stream that contains the uncompressed files. So there are 2 remaining variables in the folder that deal with this one remaining output unpacked stream:

UnPackCRC this is the CRC of this fully unpacked stream.

UnpackedStreamInfo this array stores the information about the files being split back out of this unpacked stream. In our example there are 3 files coming out of the stream, so there will be 3 entries in this array. Each entry stores the Size and CRC of the file it is representing.

Now if we go back to the beginning and find the **FileInfo** array which contains the names of the files, the entries in the **UnpackedStreamInfo** array will one to one match up to the data in the FileInfo array. Or more precisely if we remove the zero length files and the directory entries from the **FileInfo**, and then we first loop the folders and then loop the **UnpackedStreamInfo** this file data will all match up. Giving us names, sizes, and CRC's of all our non-zero length files.

This leaves us with **Codes.** The coder stores the information about the coders used inside of the folder. The Coder contains 4 data items.
The byte array **Method** stores a structured array of bytes telling us which coder should be used. This is known as the 7zip method ID. In this example LZMAs ID is 03,01,01 and BCJ2s ID is 03,03,01,1B.
Next is **NumInStreams** and **NumOutStreams** this is more or less self explanatory by now, and is the number of input streams the coder takes and the number of output streams the coder produces.
Finally there is a **Properties** byte array, some codes such as LZMA need some data to correctly pre-configure themselves to decode the supplied stream. So this array stores any setup data required by the coder being used.

Folder 1 contains a much more simple example, in folder 1 we have the following:
One **Coder** describing the LZMA coder.
The **BindPairs** array will be empty. As there are no binding pairs.
The **PackedStreamIndicies** array will have one entry with the value of 4, pointing to packed stream 4.
UnpackedStreamSizes and **UnpackedCRC** will contain the Length and CRC of the one output unpacked stream and **UnpackedStreamInfo** will contain the Size and CRC of the two output files.

The next few pages titles **7Z Data Structure** should now be fully explained.

7Z Data Structure

Header

FileInfo	FileInfo
	See FileInfo below
StreamsInfo	StreamsInfo
	See StreamsInfo Below

FileInfo

string[]	Names
	Contains the name of each file.
bool[]	EmptyStreamFlags
	If this is null then there are no empty files or directories
	If this is defined there will be one entry per file
	If true the matching entry is zero bytes entry
bool[]	EmptyfileFlags
	If this is null then all zero bytes entries are directories
	If this is defined there will be one entry per zero byte entry
	If true the matching entry is zero byte file, if false a directory

StreamsInfo

ulong	PackPosition
	Position of Streams relative to end of Signature Header
PackedStreamsInfo[]	PackedStreamsInfo
	Array of PackedStreamsInfo (See PackedStreamsInfo)
Folders[]	Folders
	Array of Folders (See Folder)

PackedStreamsInfo

ulong	PackedSize
	Length of the Stream
ulong?	Crc
	Optional CRC of the Stream
ulong	StreamPosition
	Position of the Stream Relative to StreamsInfo PackPosition

Folder

Coder[]	Coders
	Array of used Coders (See Coder)
BindPair[]	BindPairs
	Array of used BindPairs, can be empty (See BindPairs)
ulong[]	PackedStreamIndicies
	This links each packed stream to the input of the coder that will consume this stream. There is one index per packedStream. The value is an index number calculated by counting and numbering all of the NumInStreams in the Coder array.
ulong[]	UnpackedStreamSizes
	The output sizes of all of the OutStreams in the Coder array.
uint?	UnpackCRC
	The CRC of the uncompressed output stream
UnpackedStreamInfo[]	UnpackedStreamInfo
	Array of UnpackedStreamInfo, in order of FileInfo Names (non zero bytes entries)

Coder

byte[]	Method
	Byte Array hold the compression method to use
ulong	NumInStreams
	Number of Input Streams this compression method consumes
ulong	NumOutStreams
	Number of Output Streams this compression method produces
byte[]	Properties
	Byte Array used to configure the current compression method

BindPair

ulong	OutIndex
ulong	InIndex

Links a coder output to a coder input. The Input Index is from counting and numbering all of the inputs of the coders, the output Index is from counting and numbering all of the outputs of the coders

UnpackedStreamInfo

ulong	UnpackedSize
	Unpacked size of this output file
uint?	Crc
	Unpacked CRC of this output file

7z File Structure

So now on to seeing how all of this is actually stored at a byte level inside of a .7z file.

InProgress