ZLib

Introduction and Overview

The ZLib module provides a shared interface to the ZLib compression library. It provides a simple SWI interface, and a direct library replacement interface. The simple SWI interface looks the same as that for Squash will act as such. The direct replacement interface provides a number of SWIs which may be accessed as is they were the original C routines.

To complement the SWI interface, a C library is also provided as a set of veneer functions to allow the ZLib library to be called as if it had been statically linked.

The 'zlib' compression library provides lossless in-memory compression and decompression functions, including integrity checks of the uncompressed data. Compression can be done in a single step if the buffers are large enough (for example if an input file is mmap'ed), or can be done by repeated calls of the compression function. In the latter case, the caller must provide more input and/or consume the output (providing more output space) before each call.

Terminology

ZLib (q.v. RFC 1950) is a standard, freely available library produced by Jean-loup Gailly and Mark Adler which provides compression based on the standard 'deflate' algorithm (q.v. RFC 1951). In addition, the ZLib library (and module) supports the GZip compression algorithm (q.v. RFC 1952) with extensions for RISC OS file information.

Technical Details

The ZLib module provides two primary interfaces for the programmer. The first of these is a Squash-like interface to the ZLib library. Applications which already support the use of Squash can be changed to use ZLib by simply changing the SWIs that are called.

The second interface that the module provides is that of a direct replacement for the ZLib C library. Because this is provided as a SWI interface, this means that it is accessible to applications written in any language.

ZLib SWI interface

This SWIs that the module provides have equivilent (or similar) names to those of the C interface. A C interface library is provided to interact with the ZLib module such that it should not be necessary to use these calls directly from C.

Informational SWIs

The informational calls are not related to providing compression. These SWIs are the CRC32, Adler32 and Version SWIs.

GZip SWIs

The GZip calls provide GZip file compression. GZip files are compressed containers for a single file of data. GZip files contain additional information about the file they contain. This additional information is used to store RISC OS-specific file data such as the filetype, and datestamp.

ZLib SWIs

The ZLib calls provide deflate format data compression. Data compressed by the deflate algorithm is extractable by equivilent deflate decompressors. No additional data is included within the compressed data about its source. If further meta-data is required, it should be included in the output format in some application specific format. Zip files are one such example of a deflate format encapsulation.

Data formats

Stream Control Block

In order to function, a 'stream control block' must be provided to the ZLib SWIs. This control block is a private structure which can be manipulated by both the client and the ZLib module. Initially, the table should be set to 0 before being called (except where indicated).

Offset	Name	Contents
+0	next_in	Pointer to the next available input byte. This value should be zero initially, and be updated by both the client and the ZLib module as data is processed.
+4	avail_in	Amount of input data available for use by the ZLib module. This value should be zero initially, and be updated by both the client and the ZLib module as data is processed.
+8	total_out	Total number of bytes read so far.
+12	next_out	Pointer to the next output byte. This value should be zero initially, and be updated by both the client and the ZLib module as data is processed.
+16	avail_out	Amount of space in the output buffer for use by the ZLib module. This value should be zero initially, and be updated by both the client and the ZLib module as data is processed.
+20	total_out	Total number of bytes output so far.
+24	msg	Pointer to the last error message, or 0 if no error has been generated.
+28	state	Private value, controlled by the ZLib module. The client should not modify this value.
+32	zalloc	Address of memory allocator function, or 0 for ZLib to control memory allocation.
+36	zfree	Address of memory free function, or 0 for ZLib to control memory allocation.
+40	opaque	Opaque handle to pass to allocator and free functions.
+44	data_type	ZLib module's guess of the type of the data : **Value** **Meaning** 0 Binary 1 ASCII 2 Unknown
+48	adler	Adler-32 value for uncompressed data processed so far.
+52	reserved	Reserved for future expansion. Must be zero.

Flush types

Where data is being written to a stream, a 'flush type' is provided to describe what sort of operations should be performed on writing the data. Flushing may degrade compression for some compression algorithms and so it should be used only when necessary.

Type	Name	Meaning
0	Z_NO_FLUSH	Do not flush data, but just write data as normal to the output buffer. This is the normal way in which data is written to the output buffer.
1	Z_PARTIAL_FLUSH	Obsolete. You should use Z_SYNC_FLUSH instead.
2	Z_SYNC_FLUSH	All pending output is flushed to the output buffer and the output is aligned on a byte boundary, so that the decompressor can get all input data available so far.
3	Z_FULL_FLUSH	All output is flushed as with Z_SYNC_FLUSH, and the compression state is reset so that decompression can restart from this point if previous compressed data has been damaged or if random access is desired. Using Z_FULL_FLUSH too often can seriously degrade the compression. *SWI ZLib_InflateSync (on page 46)* will locate points in the compression string where a full has been performed.
4	Z_FINISH	Notifies the module that the input has now been exhausted. Pending input is processed, pending output is flushed and calls return with Z_STREAM_END if there was enough output space.

Compression levels

For compression calls, the compression level can be specified. This determines how much work is performed on trying to compress the input data. Lower compression levels indicate lesser compression, and greater speed. Higher levels indicate greater compression, but lesser speed.

Level	Name	Meaning
0	Z_NO_COMPRESSION	No compression should be used at all.
1	Z_BEST_SPEED	Minimal compression, but greatest speed.
9	Z_BEST_COMPRESSION	Maximum compression, but slowest.
-1	Z_DEFAULT_COMPRESSION	Select default compression level.

Compression strategy

For compression calls, the compression strategy can be specified. This determines what type of processing is performed on the input data. If set incorrectly, the compression strategy will produce lesser compression ratios, but does not affect the correctness of the data.

Strategy	Name	Meaning
0	Z_DEFAULT_STRATEGY	The default strategy is the most commonly used. With this strategy, string matching and huffman compression are balanced.
1	Z_FILTERED	This strategy is designed for filtered data. Data which consists of mostly small values, with random distribution should use Z_FILTERED. With this strategy, less string matching is performed.
2	Z_HUFFMAN_ONLY	This strategy performs no string matching, only the huffman encoding is performed.

Compression method

The compression method determines what algorithm is used to perform the compression. Presently only Z_DEFLATED is supported.

Method	Name	Meaning
8	Z_DEFLATED	Use deflate algorithm

Memory level

The memory level determines how much memory should be allocated for the internal compression state. The default value is 8.

Level	Meaning
1	Uses minimal memory, but is slow and reduces the compression ratio.
9	Uses maximum memory for optimal speed.

Window bits

Whilst searching for matching strings in the input data, a 'window' is used on to the previous data. This window is used to determine where matches occur. The value of the 'window bits' parameter is the base two logarithm of the size of the window. It should be in the range 9 to 15 for this version of the library. Larger values result in better compression at the expense of memory usage.

ZLib return code

Most ZLib SWIs return a 'return code'. This declares the state that the SWI encountered during the operation.

Code	Name	Meaning
0	Z_OK	No failure was encountered, the operation completed without problem.
1	Z_STREAM_END	No failure was encountered, and the input has been exhausted.
2	Z_NEED_DICT	A preset dictionary is required for the decompression of the data.
-1	Z_ERRNO	An internal error occurred
-2	Z_STREAM_ERROR	The stream structure was inconsistant
-3	Z_DATA_ERROR	Input data has been corrupted (for decompression).
-4	Z_MEM_ERROR	Memory allocation failed.
-5	Z_BUF_ERROR	There was not enough space in the output buffer.
-6	Z_VERSION_ERROR	The version supplied does not match that supported by the ZLib module.

SWI calls

ZLib_Compress (SWI &53AC0)

Simple Squash-like compression

On entry

R0 = Flags :

Bit(s)	Meaning
0	Continue previously started operation
1	More input remains after this call
2	Reserved, must be 0
3	Return workspace required
4	Workspace is not bound to an application
5-10	Reserved, must be 0

R1 - R5 = dependant on flags

On exit

R0 - R5 = dependant on flags

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is similar to *SWI Squash_Compress (on page 0)*, providing a drop in replacement for the Squash module but using ZLib compression. There are two variants of this SWI - with bit 3 set, and with bit 3 clear. Normally, this call will be made first with bit 3 set to read the workspace size, and then with bit 3 clear to perform the decompression.

Related SWIs

SWI ZLib_Decompress (on page 14)
SWI Squash_Decompress (on page 0)

ZLib_Compress bit 3 set Read workspace (SWI &53AC0)

Simple Squash-like compression

On entry

R0 = Flags :

Bit(s)	Meaning
3	Return workspace required

R1 = input size, or -1 to omit maximum output size

On exit

R0 = required workspace size
R1 = maximum output size, or -1 if it cannot be determined or was not asked for

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read the size of the buffers to use for ZLib compression via the Squash-like interface.

Related SWIs

SWI ZLib_Decompress (on page 14)
SWI Squash_Decompress (on page 0)

ZLib_Compress bit 3 clear
Compress
(SWI &53AC0)

Simple Squash-like compression

On entry

R0 = Flags :

Bit(s)	Meaning
0	Continue previously started operation
1	More input remains after this call
3	Return workspace required
4	Workspace is not bound to an application

R1 = pointer to workspace
R2 = pointer to input data
R3 = length of input data
R4 = pointer to output buffer
R5 = length of output buffer

On exit

R0 = status of decompression process :

Value	Meaning
0	Operation is complete
1	Input has been exhausted
2	Output space has been exhausted

R1 preserved
R2 = pointer to first unused byte of input data
R3 = size of unused data in input buffer
R4 = pointer to first unused byte of output buffer
R5 = size of unused data in output buffer

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to compress data to a buffer.

Related SWIs

SWI ZLib_Decompress (on page 14)
SWI Squash_Decompress (on page 0)

ZLib_Decompress (SWI &53AC1)

Simple Squash-like decompression

On entry

R0 = Flags :

Bit(s)	Meaning
0	Continue previously started operation
1	More input remains after this call
2	Assume all output will fit into the buffer
3	Return workspace required
4	Workspace is not bound to an application
5-10	Reserved, must be 0

R1 - R5 = dependant on flags

On exit

R0 - R5 = dependant on flags

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is similar to *SWI Squash_Compress (on page 0)*, providing a drop in replacement for the Squash module but using ZLib decompression. There are two variants of this SWI - with bit 3 set, and with bit 3 clear. Normally, this call will be made first with bit 3 set to read the workspace size, and then with bit 3 clear to perform the decompression.

Related SWIs

SWI ZLib_Compress (on page 9)
SWI Squash_Compress (on page 0)

ZLib_Decompress bit 3 set Read workspace (SWI &53AC1)

Simple Squash-like decompression

On entry

R0 = Flags :

Bit(s)	Meaning
3	Return workspace required

R1 = input size, or -1 to omit maximum output size

On exit

R0 = required workspace size
R1 = maximum output size, or -1 if it cannot be determined or was not asked for

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read the size of the buffers to use for ZLib decompression via the Squash-like interface.

Related SWIs

SWI ZLib_Compress (on page 9)
SWI Squash_Compress (on page 0)

ZLib_Decompress bit 3 clear Decompress (SWI &53AC1)

Simple Squash-like decompression

On entry

R0 = Flags :

Bit(s)	Meaning
0	Continue previously started operation
1	More input remains after this call
3	Return workspace required
4	Workspace is not bound to an application

R1 = pointer to workspace
R2 = pointer to input data
R3 = length of input data
R4 = pointer to output buffer
R5 = length of output buffer

On exit

R0 = status of decompression process :

Value	Meaning
0	Operation is complete
1	Input has been exhausted
2	Output space has been exhausted

R1 preserved
R2 = pointer to first unused byte of input data
R3 = size of unused data in input buffer
R4 = pointer to first unused byte of output buffer
R5 = size of unused data in output buffer

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to decompress data from a buffer.

Related SWIs

SWI ZLib_Compress (on page 9)
SWI Squash_Compress (on page 0)

ZLib_CRC32 (SWI &53AC2)

Calculate a CRC32 checksum for a given data buffer (crc32)

On entry

R0 = CRC-32 continuation value
R1 = pointer to start of block, or 0 to read the initial value to supply as a continuation value.
R2 = pointer to end of block

On exit

R0 = CRC-32 value for block
R1 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to update a running CRC-32 checksum with data from a buffer. The returned CRC-32 value should be passed back to the this SWI if the checksum needs updating further. ZLib_CRC32 is a slower call than OS_CRC, but more reliable.

Related SWIs

SWI ZLib_Adler32 (on page 20)
SWI OS_CRC

ZLib_Adler32 (SWI &53AC3)

Calculate an Adler32 checksum for a given data buffer (adler32)

On entry

R0 = Adler-32 continuation value
R1 = pointer to start of block, or 0 to read the initial value to supply as a continuation value.
R2 = pointer to end of block

On exit

R0 = Adler-32 value for block
R1 - R2 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to update a running Adler-32 checksum with data from a buffer. The returned Adler-32 value should be passed back to the this SWI if the checksum needs updating further. Adler-32 is a faster checksum to calculate than CRC-32.

Related SWIs

SWI ZLib_CRC32 (on page 19)
SWI OS_CRC

ZLib_Version (SWI &53AC4)

Return the version of ZLib in use (zlib_version)

On entry

None

On exit

R0 = pointer to read only version string

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to read the version number of the ZLib library in use.

Related APIs

None

ZLib_ZCompress (SWI &53AC5)

Compress a source buffer (compress)

On entry

R0 = pointer to output buffer for compressed data
R1 = length of output buffer
R2 = pointer to input buffer of uncompressed data
R3 = length of input buffer

On exit

R0 = ZLib return code
R1 = length of output buffer used

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to compress a block of data in a single call. The output buffer must be at least 0.1% larger than the input, plus 12 bytes.

Related SWIs

SWI ZLib_ZCompress2 (on page 23)
SWI ZLib_ZUncompress (on page 24)

ZLib_ZCompress2 (SWI &53AC6)

Compress a source buffer (compress2)

On entry

R0 = pointer to output buffer for compressed data
R1 = length of output buffer
R2 = pointer to input buffer of uncompressed data
R3 = length of input buffer
R4 = compression level

On exit

R0 = ZLib return code
R1 = length of output buffer used

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is equivilent to ZLib_Compress, but allows the compression level to be specified.

Related SWIs

SWI ZLib_ZCompress (on page 22)
SWI ZLib_ZUncompress (on page 24)

ZLib_ZUncompress (SWI &53AC7)

Compress a source buffer (uncompress)

On entry

R0 = pointer to output buffer for uncompressed data
R1 = length of output buffer
R2 = pointer to input buffer of compressed data
R3 = length of input buffer

On exit

R0 = ZLib return code
R1 = length of output buffer used

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI decompresses a buffer of data in a single call. The destination buffer must be large enough for the decompressed data.

Related SWIs

SWI ZLib_ZCompress (on page 22)

ZLib_DeflateInit (SWI &53AC8)

Initialise a stream for compression (deflateInit)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = compression level
R2 = pointer to ZLib version string expected ("1.1.4" at time of writing)
R3 = length of control block

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to initialise the ZLib compression algorithm. You should clear the workspace block to zeros. zalloc and zfree may point to routines to allocate and free memory. If these are not set then memory is allocated on a per application basis in the global dynamic area. zalloc and zfree will be entered within a C environment (ie APCS applies) with a small stack, in SVC mode. Because of this, you cannot use longjmp, use functions that require large amounts of stack space, or perform any non-SVC mode operation. Contact RISCOS Ltd if you wish to use this feature but are unsure how it will affect your code.

Related SWIs

SWI ZLib_InflateInit (on page 27)
SWI ZLib_DeflateInit2 (on page 29)
SWI ZLib_InflateInit2 (on page 31)

ZLib_InflateInit (SWI &53AC9)

Initialise a stream for decompression (inflateInit)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = pointer to ZLib version string expected ("1.1.4" at time of writing)
R2 = length of control block

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to initialise the ZLib stream decompression algorithm. You should clear the workspace block to zeros. zalloc and zfree may point to routines to allocate and free memory. If these are not set then memory is allocated on a per application basis in the global dynamic area. zalloc and zfree will be entered within a C environment (ie APCS applies) with a small stack, in SVC mode. Because of this, you cannot use longjmp, use functions that require large amounts of stack space, or perform any non-SVC mode operation. Contact RISCOS Ltd if you wish to use this feature but are unsure how it will affect your code.

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_DeflateInit2 (on page 29)
SWI ZLib_InflateInit2 (on page 31)

ZLib_DeflateInit2 (SWI &53ACA)

Initialise a stream for compression with control over parameters (deflateInit2)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = compression level
R2 = compression method
R3 = window bits for history buffer
R4 = memory level
R5 = compression strategy
R6 = pointer to ZLib version string expected ("1.1.4" at time of writing)
R7 = length of control block

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is similar to ZLib_DeflateInit, but provides much greater control than that SWI.

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_InflateInit (on page 27)
SWI ZLib_InflateInit2 (on page 31)

ZLib_InflateInit2 (SWI &53ACB)

Initialise a stream for decompression with control over parameters (inflateInit2)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = window bits
R2 = pointer to ZLib version string expected ("1.1.4" at time of writing)
R3 = length of control block

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is similar to ZLib_InflateInit, but provides much greater control than that SWI.

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_InflateInit (on page 27)
SWI ZLib_DeflateInit2 (on page 29)

ZLib_Deflate (SWI &53ACC)

Continue a stream compression (deflate)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = flush type

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI compresses as much data as possible, and stops when the input buffer becomes empty or the output buffer becomes full. `next_in` and `avail_in` are read and updated after data has been processed. You should empty the output buffer when data appears there (`next_out` and `avail_out` will have been updated).

ZLib_Deflate performs one or both of the following actions:

- Compress more input starting at `next_in` and update `next_in` and `avail_in` accordingly. If not all input can be processed (because there is not enough room in the output buffer), `next_in` and `avail_in` are updated and processing will resume at this point for the next call of ZLib_Deflate.
- Provide more output starting at `next_out` and update `next_out` and

avail_out accordingly. This action is forced if the parameter flush is non zero. Forcing flush frequently degrades the compression ratio, so this parameter should be set only when necessary (in interactive applications). Some output may be provided even if flush is not set.

Before calling ZLib_Deflate, the client should ensure that at least one of the actions is possible, by providing more input and/or consuming more output, and updating avail_in or avail_out accordingly; avail_out should never be zero before the call. The client may consume the compressed output when it wants, for example when the output buffer is full (avail_out == 0), or after each call of ZLib_Deflate. If deflate returns Z_OK and with zero avail_out, it must be called again after making room in the output buffer because there might be more output pending.

If the flush type is set to Z_SYNC_FLUSH, all pending output is flushed to the output buffer and the output is aligned on a byte boundary, so that the decompressor can get all input data available so far. In particular avail_in is zero after the call if enough output space has been provided before the call. Flushing may degrade compression for some compression algorithms and so it should be used only when necessary.

If the flush type is set to Z_FULL_FLUSH, all output is flushed as with Z_SYNC_FLUSH, and the compression state is reset so that decompression can restart from this point if previous compressed data has been damaged or if random access is desired. Using Z_FULL_FLUSH too often can seriously degrade the compression.

If deflate returns with avail_out == 0, this function must be called again with the same value of the flush parameter and more output space (updated avail_out), until the flush is complete (ZLib_Deflate returns with non-zero avail_out).

If the flush type is set to Z_FINISH, pending input is processed, pending output is flushed and ZLib_Deflate returns with Z_STREAM_END if there was enough output space; if ZLib_Deflate returns with Z_OK, this function must be called again with Z_FINISH and more output space (updated avail_out) but no more input data, until it returns with Z_STREAM_END or an error. After ZLib_Deflate has returned Z_STREAM_END, the only possible operations on the stream are *SWI ZLib_DeflateReset (on page 43)* or *SWI ZLib_DeflateEnd (on page 35).*

Z_FINISH can be used immediately after *SWI ZLib_DeflateInit (on page 25)* if all the compression is to be done in a single step. In this case, avail_out must be at least 0.1% larger than avail_in plus 12 bytes. If ZLib_Deflate does not return Z_STREAM_END, then it must be called again as described above.

ZLib_Deflate sets `adler` to the adler32 checksum of all input read so far (that is, `total_in` bytes).

ZLib_Deflate may update `data_type` if it can make a good guess about the input data type (Z_ASCII or Z_BINARY). In doubt, the data is considered binary. This field is only for information purposes and does not affect the compression algorithm in any manner.

ZLib_Deflate returns Z_OK if some progress has been made (more input processed or more output produced), Z_STREAM_END if all input has been consumed and all output has been produced (only when the flush type is set to Z_FINISH), Z_STREAM_ERROR if the stream state was inconsistent (for example if `next_in` or `next_out` was NULL), Z_BUF_ERROR if no progress is possible (for example `avail_in` or `avail_out` was zero).

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_DeflateInit2 (on page 29)
SWI ZLib_DeflateEnd (on page 35)

ZLib_DeflateEnd (SWI &53ACD)

Terminate a stream compression (deflateEnd)

On entry

R0 = pointer to *#subsubsection_stream_control_block*

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI frees all the memory used by the compression algorithm, discarding any unprocessed input or output.

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_DeflateInit2 (on page 29)
SWI ZLib_Deflate (on page 32)

ZLib_Inflate (SWI &53ACE)

Continue decompressing a stream (inflate)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = flush type

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI decompresses as much data as possible, and stops when the input buffer becomes empty or the output buffer becomes full. `next_in` and `avail_in` are read and updated after data has been processed. You should empty the output buffer when data appears there (`next_out` and `avail_out` will have been updated).

ZLib_Inflate performs one or both of the following actions:

- Decompress more input starting at next_in and update next_in and avail_in accordingly. If not all input can be processed (because there is not enough room in the output buffer), next_in is updated and processing will resume at this point for the next call of ZLib_Inflate.
- Provide more output starting at `next_out` and update `next_out` and avail_out accordingly. ZLib_Inflate provides as much output as

possible, until there is no more input data or no more space in the output buffer (see below about the flush parameter).

Before the call of ZLib_Inflate, the application should ensure that at least one of the actions is possible, by providing more input and/or consuming more output, and updating the `next_in`/`next_out` and `avail_in`/`avail_out` values accordingly. The application can consume the uncompressed output when it wants, for example when the output buffer is full (`avail_out` == 0), or after each call of ZLib_Inflate. If this SWI returns Z_OK and with zero `avail_out`, it must be called again after making room in the output buffer because there might be more output pending.

If R1 is set to Z_SYNC_FLUSH, ZLib_Inflate flushes as much output as possible to the output buffer. The flushing behavior of inflate is not specified for values of the flush parameter other than Z_SYNC_FLUSH and Z_FINISH, but the current implementation actually flushes as much output as possible anyway.

ZLib_Inflate should normally be called until it returns Z_STREAM_END or an error. However if all decompression is to be performed in a single step (a single call of ZLib_Inflate), the parameter flush should be set to Z_FINISH. In this case all pending input is processed and all pending output is flushed ; `avail_out` must be large enough to hold all the uncompressed data. (The size of the uncompressed data may have been saved by the compressor for this purpose.) The next operation on this stream must be inflateEnd to deallocate the decompression state. The use of Z_FINISH is never required, but can be used to inform inflate that a faster routine may be used for the single ZLib_Inflate call.

If a preset dictionary is needed at this point (see *#swi_zlib_inflatesetdictionary* below), ZLib_Inflate sets `adler` to the Adler-32 checksum of the dictionary chosen by the compressor and returns Z_NEED_DICT ; otherwise it sets `adler` to the Adler-32 checksum of all output produced so far (that is, total_out bytes) and returns Z_OK, Z_STREAM_END or an error code as described below. At the end of the stream, ZLib_Inflate checks that its computed Adler-32 checksum is equal to that saved by the compressor and returns Z_STREAM_END only if the checksum is correct.

ZLib_Inflate returns Z_OK if some progress has been made (more input processed or more output produced), Z_STREAM_END if the end of the compressed data has been reached and all uncompressed output has been produced, Z_NEED_DICT if a preset dictionary is needed at this point, Z_DATA_ERROR if the input data was corrupted (input stream not conforming to the ZLib format or incorrect Adler-32 checksum), Z_STREAM_ERROR if the stream structure was inconsistent (for example if next_in or next_out was NULL), Z_MEM_ERROR if there was not enough memory, Z_BUF_ERROR if no progress is possible or if there was not enough room in the output buffer when Z_FINISH is used. In the Z_DATA_ERROR case, the application may then call *SWI ZLib_InflateSync (on page 46)* to look for a good compression block.

Related SWIs

SWI ZLib_InflateInit (on page 27)
SWI ZLib_InflateInit2 (on page 31)
SWI ZLib_InflateEnd (on page 39)

ZLib_InflateEnd (SWI &53ACF)

Terminate a decompression stream (inflateEnd)

On entry

R0 = pointer to *#subsubsection_stream_control_block*

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI frees all the memory used by the decompression algorithm, discarding any unprocessed input or output.

Related SWIs

SWI ZLib_InflateInit (on page 27)
SWI ZLib_InflateInit2 (on page 31)
SWI ZLib_Inflate (on page 36)

ZLib_DeflateSetDictionary (SWI &53AD0)

Initialise a string dictionary for a stream compression (deflateSetDictionary)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = pointer to dictionary block (a stream of bytes)
R2 = length of dictionary block

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to initialise a compression dictionary. The dictionary consists of strings (byte sequences) that are likely to be encountered later in the data to be compressed, with the most commonly used strings preferably put towards the end of the dictionary.

The dictionary should consist of strings (byte sequences) that are likely to be encountered later in the data to be compressed, with the most commonly used strings preferably put towards the end of the dictionary. Using a dictionary is most useful when the data to be compressed is short and can be predicted with good accuracy ; the data can then be compressed better than with the default empty dictionary.

Depending on the size of the compression data structures selected by ZLib_DeflateInit or ZLib_DeflateInit2, a part of the dictionary may in effect be discarded, for example if the dictionary is larger than the window size in ZLib_Deflate or ZLib_Deflate2. Thus the strings most likely to be useful should be put at the end of the dictionary, not at the front.

Upon return of this function, `adler` is set to the Adler-32 value of the dictionary; the decompressor may later use this value to determine which dictionary has been used by the compressor. (The Adler32 value applies to the whole dictionary even if only a subset of the dictionary is actually used by the compressor.)

ZLib_DeflateSetDictionary returns Z_OK if success, or Z_STREAM_ERROR if a parameter is invalid (such as NULL dictionary) or the stream state is inconsistent (for example if ZLib_Deflate has already been called for this stream). ZLib_DeflateSetDictionary does not perform any compression: this will be done by ZLib_Deflate.

Related SWIs

SWI ZLib_DeflateInit (on page 25)

ZLib_DeflateCopy (SWI &53AD1)

Copy the compression state (deflateCopy)

On entry

R0 = pointer to destination *#subsubsection_stream_control_block*
R1 = pointer to source *#subsubsection_stream_control_block*

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to take a copy of the current compression state. This might be useful if you are attempting to filter the data in one of a number of ways.

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_DeflateReset (on page 43)

ZLib_DeflateReset (SWI &53AD2)

Reset the internal compression state (deflateReset)

On entry

R0 = pointer to *#subsubsection_stream_control_block*

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is equivilent to ZLib_DeflateEnd followed by ZLib_DeflateInit.

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_DeflateEnd (on page 35)

ZLib_DeflateParams (SWI &53AD3)

Modifies the compression parameters (deflateParams)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = compression level
R2 = compression strategy

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is updates the compression level and strategy. You may do this part way through compression.

Related SWIs

SWI ZLib_DeflateInit (on page 25)
SWI ZLib_DeflateInit2 (on page 29)

ZLib_InflateSetDictionary (SWI &53AD4)

Initialise a string dictionary for a decompression stream (inflateSetDictionary)

On entry

R0 = pointer to *#subsubsection_stream_control_block*
R1 = pointer to dictionary block (a stream of bytes)
R2 = length of dictionary block

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to initialise a decompression dictionary. The dictionary must be the same as that used to compress the data (ZLib_DeflateSetDictionary).

Related SWIs

SWI ZLib_DeflateSetDictionary (on page 40)

ZLib_InflateSync (SWI &53AD5)

Re-synchronise decompression stream (inflateSetDictionary)

On entry

R0 = pointer to *#subsubsection_stream_control_block*

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI will skip invalid data until a full flush point is found. This may be useful if you have data which is likely to be corrupted, is periodically synchronised and the data 'lost' is unimportant. An example might be a stream of graphical or audio data.

Related APIs

None

ZLib_InflateReset (SWI &53AD6)

Reset the decompression stream state (inflateReset)

On entry

R0 = pointer to *#subsubsection_stream_control_block*

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is equivilent to ZLib_InflateEnd followed by ZLib_InflateInit.

Related SWIs

SWI ZLib_InflateInit (on page 27)

ZLib_GZOpen (SWI &53AD7)

Open a GZip file for reading or writing (gzopen)

On entry

R0 = pointer to filename
R1 = pointer to the 'mode' of file operation. This consists of two parts; the access type and the modifiers. Only one access type may be used, but multiple modifiers may be added to the end.

Access type	Meaning
rb	Open for reading
wb	Open for writing

Modifier	Meaning
0-9	compression level
h	Huffman compression only
f	Data is 'filtered' (small values, randomly distributed)
f	RISC OS type information attached

R2 = load address of file (if 'R' and 'wb' used)
R3 = exec address of file (if 'R' and 'wb' used)
R4 = length address of file (if 'R' and 'wb' used)
R5 = attributes address of file (if 'R' and 'wb' used)

On exit

R0 = opaque GZip handle
R2 = load address of file (if 'R' and 'rb' used)
R3 = exec address of file (if 'R' and 'rb' used)
R4 = length address of file (if 'R' and 'rb' used)
R5 = attributes address of file (if 'R' and 'rb' used)

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI opens a file for accessing GZip compressed data. The 'R' extension is intended for compressing RISC OS files completely losslessly. Expanding such files on other systems will result in the loss of RISC OS type information only; the file data itself will be intact.

Related SWIs

SWI ZLib_GZRead (on page 50)
SWI ZLib_GZWrite (on page 51)
SWI ZLib_GZFlush (on page 52)
SWI ZLib_GZClose (on page 53)
SWI ZLib_GZSeek (on page 55)
SWI ZLib_GZTell (on page 57)
SWI ZLib_GZEOF (on page 58)

ZLib_GZRead (SWI &53AD8)

Read data from a GZip file (gzread)

On entry

R0 = opaque GZip handle
R1 = pointer to destination buffer
R2 = amount of data to read

On exit

R0 = number of bytes read

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI reads data from a previously opened GZip file.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_GZWrite (on page 51)

ZLib_GZWrite (SWI &53AD9)

Write data to a GZip file (gzwrite)

On entry

R0 = opaque GZip handle
R1 = pointer to source buffer
R2 = amount of data to write

On exit

R0 = number of bytes written

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI writes data to a previously opened GZip file.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_GZRead (on page 50)

ZLib_GZFlush (SWI &53ADA)

Flush all pending data to a GZip file (gzflush)

On entry

R0 = opaque GZip handle
R1 = flush type

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI writes data to a previously opened GZip file.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_Deflate (on page 32)

ZLib_GZClose (SWI &53ADB)

Close a GZip file (gzclose)

On entry

R0 = opaque GZip handle

On exit

R0 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI closes a previously opened GZip file.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_GZRead (on page 50)

ZLib_GZError (SWI &53ADC)

Close a GZip file (gzclose)

On entry

R0 = opaque GZip handle

On exit

R0 = pointer to last error message string
R1 = ZLib return code

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI returns the last error message returned by a GZip operation.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_GZRead (on page 50)
SWI ZLib_GZWrite (on page 51)
SWI ZLib_GZFlush (on page 52)
SWI ZLib_GZSeek (on page 55)

ZLib_GZSeek (SWI &53ADD)

Move to a specific location in a GZip file (gzseek)

On entry

R0 = opaque GZip handle
R1 = position in decompressed data in bytes
R2 = type of seek to perform :

Type	Meaning
0	Set absolute position (position = R1)
1	Set relative position (position = R1 + current position)

On exit

R0 = new position in file

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI changes the pointer within a GZip file. The position specified is located such that byte number R1 had just been read from the file. The operation cannot be performed on files being written.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_GZRead (on page 50)
SWI ZLib_GZTell (on page 57)

ZLib_GZTell (SWI &53ADE)

Return the current position in a GZip file (gztell)

On entry

R0 = opaque GZip handle

On exit

R0 = current position in decompressed data in bytes

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI returns the current position in the decompressed data in bytes as an offset from the start of the data.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_GZRead (on page 50)
SWI ZLib_GZSeek (on page 55)

ZLib_GZEOF (SWI &53ADF)

Check whether the end of file has been reached (gztell)

On entry

R0 = opaque GZip handle

On exit

R0 = 1 if EOF condition has been reached, 0 otherwise

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI checks whether the end of the file has been reached.

Related SWIs

SWI ZLib_GZOpen (on page 48)
SWI ZLib_GZRead (on page 50)

ZLib_TaskAssociate (SWI &53AE0)

Change Wimp Task association for a stream

On entry

R0 = pointer to *#subsubsection_stream_control_block*

On exit

R0 preserved

Interrupts

Interrupts are disabled
Fast interrupts are enabled

Processor mode

Processor is in SVC mode

Re-entrancy

SWI is not re-entrant

Use

This SWI is used to associate a ZLib stream with a task. When associated, the tasks death will cause the memory allocated to the stream to be released automatically. If dissociated, the stream will never be freed unless the deflateEnd or inflateEnd calls are issued.

By default all streams are associated with the task with which they were created and destroyed automatically should that task terminate. You may wish to disable this operation using this SWI.

Related APIs

None

Document information

Maintainer(s):	Charles Ferguson <gerph@gerph.org>			
History:	**Revision**	**Date**	**Author**	**Changes**
	1		Gerph	Initial version
	2		Gerph	Updates for 1.1.4
Disclaimer:	© Gerph.			