Welcome to the aPLib documentation!

aPLib is a compression library based on the algorithm used in aPACK (my 16-bit executable packer). aPLib is an easy-to-use alternative to many of the heavy-weight compression libraries available.

The compression ratios achieved by aPLib combined with the speed and tiny footprint of the decompressors (as low as 169 bytes!) makes it the ideal choice for many products.

Contents:

License

aPLib is freeware. If you use aPLib in a product, an acknowledgement would be appreciated, e.g. by adding something like the following to the documentation:

> This product uses the aPLib compression library,
> Copyright © 1998-2014 Joergen Ibsen, All Rights Reserved.
> For more information, please visit:
> http://www.ibsensoftware.com/

You may not redistribute aPLib without all of the files.

You may not edit or reverse engineer any of the files (except the header files and the decompression code, which you may edit as long as you do not remove the copyright notice).

You may not sell aPLib, or any part of it, for money (except for charging for the media).

```
#ifndef COMMON_SENSE
```

> This software is provided “as is”. In no event shall I, the author, be liable for any kind of loss or damage arising out of the use, abuse or the inability to use this software. USE IT ENTIRELY AT YOUR OWN RISK!
>
> This software comes without any kind of warranty, either expressed or implied, including, but not limited to the implied warranties of merchantability or fitness for any particular purpose.
>
> If you do not agree with these terms or if your jurisdiction does not allow the exclusion of warranty and liability as stated above you are NOT allowed to use this software at all.

```
#else
```

Bla bla bla .. the usual stuff - you know it anyway:

If anything goes even remotely wrong - blame _yourself_, NOT me!

```
#endif
```

General Information

Introduction

aPLib is a compression library based on the algorithm used in aPACK (my 16-bit executable packer). aPLib is an easy-to-use alternative to many of the heavy-weight compression libraries available.

The compression ratios achieved by aPLib combined with the speed and tiny footprint of the decompressors (as low as 169 bytes!) makes it the ideal choice for many products.

Since the first public release in 1998, aPLib has been one of the top pure LZ-based compression libraries available. It is used in a wide range of products including executable compression and protection software, archivers, games, embedded systems, and handheld devices.

Compatibility

The aPLib package includes pre-compiled libraries in a number of formats (COFF, ELF, OMF).

No standard library functions are used, so the libraries can work with most x86/x64 compilers, as long as the name decoration and calling conventions match.

The ELF folders contain a version of aPLib which uses PIC to allow it to be linked into shared libraries on linux.

Thread-safety

All compression and decompression functions are thread-safe.

Using aPLib

For C/C++ you simply include `aplib.h` and link with the appropriate library for your compiler. If you only need to decompress data, or if you modify the decompression code, you can compile and link with one of the decompression implementations in the `src` folder.

For other languages you can either check if there is a useable example, or use the DLL version. Most linkers allow calling C functions in an external library, so usually there is a way to use one of the libraries.

aPLib performs memory-to-memory compression and decompression, so getting data into an input buffer and allocating an output buffer is your responsibility.

All functions return `APLIB_ERROR` (which is `-1`) if an error occurs.

Attempting to compress incompressible data can lead to expansion. You can get the maximum possible coded size by passing the size of the input to the function **`aP_max_packed_size()`**.

When calling **`aP_pack()`** you have to supply a work buffer. You can get the required size of this buffer by passing the size of the input to the function **`aP_workmem_size()`** (in the current version this function always returns 640k).

If you do not have a callback for **`aP_pack()`**, you can pass `NULL` (i.e. `0`) instead. The callback functionality allows your program to keep track of the compression progress, and if required to stop the compression.

`aP_depack()`, **`aP_depack_asm()`**, and **`aP_depack_asm_fast()`** assume that they are given valid compressed data – if not they will most likely crash. This is to ensure that the basic decompression code is as small, fast and easy to understand as possible. You can use **`aP_depack_safe()`** or **`aP_depack_asm_safe()`** if you need to catch decompression errors. Also the safe wrapper functions provide a nice interface that helps prevent potential crashes.

Safe Wrapper Functions

Starting with aPLib v0.34, there are additional functions included which provide a better way of handling the compressed data in the example, and also serve as an example of how to add functionality through function wrappers.

The **`aPsafe_pack()`** and **`aPsafe_depack()`** functions are wrappers for their regular `aP_` counterparts, which add a header to the compressed data. This header includes a tag, information about the compressed and decompressed size of the data, and CRC32 values for the compressed and decompressed data.

The `example` folder contains a simple command line packer that uses aPLib to compress and decompress data. The `aPsafe_` functions are used in this example, because they provide extra functionality like retrieving the original size of compressed data.

Contact Information

If you have any questions, suggestions or bug-reports about aPLib, please feel free to contact me by e-mail at:

> contact@ibsensoftware.com

You can get the latest version of aPLib and my other software at:

> http://www.ibsensoftware.com/

Compression

The following is a description of the aPLib compression functionality.

Compression Functions

size_t **`aP_pack`**(const void **source*, void **destination*, size_t *length*, void **workmem*, int *(*callback)(size_t*, size_t, size_t, void **)*, void **cbparam*)

Compress *length* bytes of data from *source* to *destination*, using *workmem* as temporary storage.

The *destination* buffer should be large enough to hold `aP_max_packed_size(length)` bytes.

The *workmem* buffer should be `aP_workmem_size(length)` bytes large.

The callback function, *callback*, must take four parameters. The first is *length*, the second is the number of input bytes that has been compressed, the third is how many output bytes they have been compressed to, and the fourth is *cbparam*. If you do not have a callback, use `NULL` instead. If the callback returns a non-zero value then **`aP_pack()`** will continue compressing – if it returns zero, **`aP_pack()`** will stop and return `APLIB_ERROR`.

Parameters:	• **source** – pointer to data to be compressed • **destination** – pointer to where compressed data should be stored • **length** – length of uncompressed data in bytes • **workmem** – pointer to work memory used during compression • **callback** – pointer to callback function (or `NULL`) • **cbparam** – callback argument
Returns:	length of compressed data, or `APLIB_ERROR` on error

size_t **`aP_workmem_size`**(size_t *input_size*)

Compute required size of *workmem* buffer used by `aP_pack()` for compressing *input_size* bytes of data.

The current code always returns 640k (640*1024).

Parameters:	• **input_size** – length of uncompressed data in bytes
Returns:	required length of work buffer

size_t **`aP_max_packed_size`**(size_t *input_size*)

Compute maximum possible compressed size when compressing *input_size* bytes of incompressible data.

The current code returns `(input_size + (input_size / 8) + 64)`.

Parameters:	• **input_size** – length of uncompressed data in bytes
Returns:	maximum possible size of compressed data

Safe Wrapper Functions

size_t **`aPsafe_pack`**(const void **source*, void **destination*, size_t *length*, void **workmem*, int *(*callback)(size_t*, size_t, size_t, void **)*, void **cbparam*)

Wrapper function for `aP_pack()`, which adds a header to the compressed data containing the length of the original data, and CRC32 checksums of the original and compressed data.

Parameters:	• **source** – pointer to data to be compressed • **destination** – pointer to where compressed data should be stored • **length** – length of uncompressed data in bytes • **workmem** – pointer to work memory used during compression • **callback** – pointer to callback function (or `NULL`) • **cbparam** – callback argument
Returns:	length of compressed data, or `APLIB_ERROR` on error

See also: `aP_pack()`

Example

```
/* allocate workmem and destination memory */
char *workmem    = malloc(aP_workmem_size(length));
char *compressed = malloc(aP_max_packed_size(length));

/* compress data[] to compressed[] */
size_t outlength = aPsafe_pack(data, compressed, length, workme

/* if APLIB_ERROR is returned, and error occured */
if (outlength == APLIB_ERROR) {
        printf("An error occured!\n");
}
else {
        printf("Compressed %u bytes to %u bytes\n", length, out
}
```

Decompression

The following is a description of the aPLib decompression functionality.

Decompression Functions

size_t **`aP_depack`**(const void **source*, void **destination*)

Decompress compressed data from *source* to *destination*.

The *destination* buffer must be large enough to hold the decompressed data.

Parameters:	• **source** – pointer to compressed data • **destination** – pointer to where decompressed data should be stored
Returns:	length of decompressed data, or `APLIB_ERROR` on error

Note: This function is not included in the libraries, but is available in `src/c/depack.c`. **`aP_depack_asm_fast()`** can be used instead.

size_t **`aP_depack_safe`**(const void **source*, size_t *srclen*, void **destination*, size_t *dstlen*)

Decompress compressed data from *source* to *destination*.

This function reads at most *srclen* bytes from *source*, and writes at most *dstlen* bytes to *destination*. If there is not enough source or destination space, or a decoding error occurs, the function returns `APLIB_ERROR`.

Parameters:	• **source** – pointer to compressed data • **srclen** – size of source buffer in bytes • **destination** – pointer to where decompressed data should be stored • **dstlen** – size of destination buffer in bytes
	length of decompressed data, or `APLIB_ERROR` on

Returns:	error

Note: This function is not included in the libraries, but is available in `src/c/depacks.c`. `aP_depack_asm_safe()` can be used instead.

size_t **`aP_depack_asm`**(const void **source*, void **destination*)

Decompress compressed data from *source* to *destination*.

The *destination* buffer must be large enough to hold the decompressed data.

Optimised for size.

Parameters:	• **source** – pointer to compressed data • **destination** – pointer to where decompressed data should be stored
Returns:	length of decompressed data, or `APLIB_ERROR` on error

size_t **`aP_depack_asm_fast`**(const void **source*, void **destination*)

Decompress compressed data from *source* to *destination*.

The *destination* buffer must be large enough to hold the decompressed data.

Optimised for speed.

Parameters:	• **source** – pointer to compressed data • **destination** – pointer to where decompressed data should be stored
Returns:	length of decompressed data, or `APLIB_ERROR` on error

size_t **`aP_depack_asm_safe`**(const void **source*, size_t *srclen*, void **destination*, size_t *dstlen*)

Decompress compressed data from *source* to *destination*.

This function reads at most *srclen* bytes from *source*, and writes at most *dstlen* bytes to *destination*. If there is not enough source or destination space, or a decoding error occurs, the function returns `APLIB_ERROR`.

Parameters:	• **source** – pointer to compressed data • **srclen** – size of source buffer in bytes • **destination** – pointer to where decompressed data should be stored • **dstlen** – size of destination buffer in bytes
Returns:	length of decompressed data, or `APLIB_ERROR` on error

See also: `aPsafe_depack()`

unsigned int **`aP_crc32`**(const void **source*, size_t *length*)

Compute CRC32 value of *length* bytes of data from *source*.

Parameters:	• **source** – pointer to data to process • **length** – size in bytes of data
Returns:	CRC32 value

Safe Wrapper Functions

size_t **`aPsafe_check`**(const void **source*)

Compute CRC32 of compressed data in *source* and check it against value stored in header. Return length of decompressed data stored in header.

Parameters:	• **source** – compressed data to process
Returns:	length of decompressed data, or `APLIB_ERROR` on error

size_t **`aPsafe_get_orig_size`**(const void **source*)

Return length of decompressed data stored in header of compressed data in *source*.

Parameters:	• **source** – compressed data to process
Returns:	length of decompressed data, or `APLIB_ERROR` on error

size_t **`aPsafe_depack`**(const void **source*, size_t *srclen*, void **destination*, size_t *dstlen*)

Wrapper function for `aP_depack_asm_safe()`, which checks the CRC32 of the compressed data, decompresses, and checks the CRC32 of the decompressed data.

Parameters:	• **source** – pointer to compressed data • **srclen** – size of source buffer in bytes • **destination** – pointer to where decompressed data should be stored • **dstlen** – size of destination buffer in bytes
Returns:	length of decompressed data, or `APLIB_ERROR` on error

See also: `aP_depack_asm_safe()`

Example

```
/* get original size */
size_t orig_size = aPsafe_get_orig_size(compressed);

/* allocate memory for decompressed data */
char *data = malloc(orig_size);

/* decompress compressed[] to data[] */
size_t outlength = aPsafe_depack(compressed, compressed_size, d

/* check decompressed length */
if (outlength != orig_size) {
        printf("An error occured!\n");
}
else {
        printf("Decompressed %u bytes\n", outlength);
}
```

Acknowledgements

Greetings and thanks to:

- d’b for our continuous discussions of compression techniques :)
- TAD for all the great ideas and the good discussions
- The people who made the Epsilon Compression Page
- Pasi ‘Albert’ Ojala for his info on PuCrunch
- RIT Research Labs for making Dos Navigator .. it’s the BEST!
- LiuTaoTao for making TR .. one of the best debuggers around!
- Eugene Suslikov (SEN) for making HIEW .. it ROCKS!
- Oleg for his work on the TMT Pascal code
- Veit Kannegieser for his work on the VPascal code
- METALBRAIN for his work on the 16bit depackers
- Gautier for his work on the Ada code
- Alexey Solodovnikov for his work on the Delphi code
- Steve Hutchesson for his work on the MASM32 code
- Agner Fog for objconv and his great info on calling conventions
- All other people who make good software freely available for non-commercial use!

A special thanks to the beta-testers:

- x-otic (thx mate ;)
- Oleg Prokhorov (great optimisations and bug reports!)
- Lawrence E. Boothby
- METALBRAIN (believe in miracles, my friend ;)
- eL PuSHeR
- Elli
- Veit Kannegieser
- Gautier

Version History

v1.1.1 *

Add `VERSIONINFO` resource to dll files, and fix subsystem in 32-bit dll for Win95 compatibility, thanks to Richard Russell.

Add Apple II example, thanks to Peter Ferrie.

Add PowerBASIC example, thanks to Wayne Diamond.

Use Sphinx to generate docs.

v1.1.0 *

Added Linux ELF shared library support, thanks to Vov Pov.

Added a Python example, thx to Marco Fabbricatore.

Fixed a bug in 64-bit **`aPsafe_check()`**.

Cleaned up compression code.

Changed to semver version numbering.

v1.01 *

Added undecorated names to 32-bit dll again, thanks to James C. Fuller.

v1.00 *

Changed the license so aPLib can now be used free of charge for commercial use as well.

Added support for 64-bit compression and decompression. Since I do not have a running 64-bit system myself, any feedback on how it works would be great.

Removed support for a number of old compilers/assemblers. If you still need these, please use the previous release or contact me (if you need object files for Delphi, simply unpack the OMF library).

Moved most of the examples to the contrib folder since I no longer have the old development tools installed to check they work.

All the assembly source files included are now FASM syntax.

Simplified the build process using objconv by Agner Fog.

Jumped the version number to v1.00 to signify the code is stable.

v0.44 *

Made a few updates to the documentation.

Fixed a rare crash, thx to Rafael Ahucha!

v0.43 *

Added Visual Basic 6 wrapper, thx to Jon Johnson!

Added PowerBuilder 9.0 objects, thx to James Sheekey!

Fixed a rare crash, thx to cyberbob!

v0.42 *

Added C and assembler implementations of a new safe depacker **`aP_depack_safe()`** and **`aP_depack_asm_safe()`**.

Updated the `aPsafe_` wrapper functions.

Renamed `lib/vc` to `lib/mscoff` and `lib/watcom` to `lib/omf` to better reflect that they are not limited to those specific compilers.

Updated examples and documentation.

v0.41

Added a Borland C++ Builder example, thx to mouser!

Fixed vc library compatibility with Pelles C.

v0.40

The documentation was rewritten in html, and moved to a separate folder.

All examples were updated. The `dll_asm`, `dos32` and `tlink32` examples were removed, and a small .NET example was added.

v0.39

All aPLib functions now return `-1` on error instead of `0`. Added a macro `APLIB_ERROR` for this value to all include files.

v0.38

The aPLib compression functions should now be fully thread-safe. Updated the C decompression code for thread-safety.

v0.37

Changed the parameters for the callback function. It is now called with the input size, input bytes processed, output bytes produced, and a user-supplied callback parameter. Thx to f0dder!

v0.36 *

Fixed a bug which could cause a match to be found in the area before the input buffer under certain conditions, thx to Veit!

Changed the extension of the C example files from `cpp` to `c`.

The ELF32 version was tested under FreeBSD, thx to Oleg!

v0.35

Worked with a number of ‘issues’ in the build process.

Fixed the C depacker so it no longer modifies the input buffer, thx to Trevor Mensah!

v0.34 *

Updated the 16bit, Ada, Delphi, C/C++, TMT Pascal and Virtual Pascal examples, thx to METALBRAIN, Gautier, Oleg and Veit!

Added a MASM32 example program, thx to Steve Hutchesson!

Fixed another bug which could cause `aP_pack()` to read one byte past the input buffer, thx to Reiner Proels!

NOTE!! the dll version now expects the callback function to use the stdcall calling convention.

The libraries now include the function `aP_max_packed_size()`, which given the input size returns the maximum possible size `aP_pack()` may produce (i.e. the worst case output size of totally incompressible data). At the moment the function simply returns `(inputsize + (inputsize / 8) + 64)`.

v0.33

Added ELF32 version of aPLib, which has been tested with Linux, BeOS and QNX. Modified the C example to work under these operating systems too.

v0.32

Discovered some mixups between different versions of the examples .. started rewriting some of them. Added a header to the files created by most of the examples.

v0.31

Improved compression ratio a little.

v0.30

Fixed a bug in one of the 16bit depackers, thx to Peter Hegel!

Updated the C/C++ example.

v0.29

Updated the Ada example, thx to Gautier!

I have removed the ‘b’ from the version number.

v0.28b

Updated the 16bit depacker examples, thx to METALBRAIN!

Renamed the SRC/C depacker files.

v0.27b

Fixed a bug which could cause `aP_pack()` to read one byte past the input buffer.

v0.26b *

Added Visual C++ and Borland C examples.

Rewrote the example program, so there is only a single source file, which works with BCC32, DJGPP, VC++ and Watcom.

Added an import library for Visual C++ in `lib/dll`, and an example of how to use it (`examples/c/make_dll.bat`).

The libraries now include the function `aP_workmem_size()`, which given the input size returns the amount of memory required for the work buffer (you still have to allocate it yourself). This should make upgrading easier in case I change the memory requirement in a later version. At the moment the function simply returns 640k.

v0.25b

Added a TMT Pascal example, thx to Oleg Prokhorov!

Moved the Ada and VPascal examples to the example dir.

Updated the documentation.

v0.24b

Updated the 16bit depacker examples, thx to METALBRAIN!

Made all assembler depackers smaller, thx to TAD and METALBRAIN!

v0.23b

Recompiled with the latest VC++ and DJGPP versions.

Did a few speed optimisations – most versions should be a little faster.

v0.22b *

Improved the compression speed a little more.

Cleaned up the code, which made the library somewhat smaller.

Added a C depacker.

Silent update: Updated the 16bit depackers - thx to METALBRAIN. There is still one problem with the 16bit example depackers, but it will be fixed for the next release.

v0.21b

Improved compression ratio and speed.

Added Ada support by Gautier - thx!

Reduced the memory requirement from 1mb to 640k – which should be enough for anybody ;).

v0.20b *

Added Delphi support and example by Alexey Solodovnikov - thx!

Rewrote the aPPack example, removing some errors, and added 16bit depackers - thx to METALBRAIN!

Removed a lot of unneeded information from the object files.

Rearranged all the folders – hope it's not too confusing ;).

Removed all the example binaries from this file, and made them available in a separate file instead.

Added the real aPACK / aPLib homepage URL, since home.ml.org was down for a period.

v0.19b *

Fixed a little mem bug (hopefully), thx to ANAKiN!

Ratio improved a little on large files.

Finally got around to updating my DJGPP installation :)

NOTE!! I have revised the license conditions – please read APLIB.DOC.

v0.18b *

Added the new VPascal interfacing code by Veit Kannegieser.

Added a library compiled for VC.

Worked a little on the depackers.

v0.17b *

NOTE!! the callback function now has to return a value. This is to make it possible for the callback function to abort the packing without exiting the program. If the callback returns `1`, **`aP_pack()`** will continue – if it returns `0`, **`aP_pack()`** will stop and return `0`.

The aPACK / aPLib homepage is now up on: apack.home.ml.org

I have not gotten the new VPascal interfacing code from Veit yet, so I will add it again in the next version :)

Since I have added so much new stuff, I am releasing this version to get some feedback (hint!), to find out where to go from here. If you have Visual C++, Borland C++, Borland C++ Builder, Visual Basic, Delphi or other 32-bit compilers/linkers, I am very interested in any problems you might have using aPLib (especially the DLL version).

v0.16b

NOTE!! `aP_pack()` NO LONGER allocates the memory it needs itself. This was changed because otherwise you would need to supply malloc and free functions to the packer. Now you just call `aP_pack()` with a pointer to 1mb of mem. This is also faster if you compress multiple sets of data, because mem is not allocated and deallocated every time.

Added new VPascal interfacing code by Veit Kannegieser - thx!

Added DOS32 and TLINK32 (Win32 PE) example code and executables.

Also added a DLL version of aPLib, and some example code for it. By the way – the DLL version works fine as a wdl file for WDOSX!

Speeded up the fast depackers a tiny bit.

v0.15b

Quite a few people have pointed out to me that AR was not the cleverest library format to use, so I changed to OMF format, which works with (at least) Watcom, DOS32 and TASM32/TLINK32.

Added assembler depackers for TASM and WASM, and added the fast assembler depacker for NASM.

The Watcom and DJGPP libraries now also contain compiled versions of `aP_depack_asm()` and `aP_depack_asm_fast()`, and the `APDEPACK.H` files with the inline assembler versions have been removed.

v0.14b *

Made some minor enhancements to the packer - ratio is a little better.

Added depacking code for NASM, converted by Archee/CoNTRACT - thx!

v0.13b

Added depacking code for Pascal (Virtual Pascal), converted by Veit Kannegieser - thx!

Switched to AR format for the Watcom library.

v0.12b *

Changed the libraries to make them C-compatible.

v0.11b

`aP_depack_asm_fast()` is a little faster.

v0.10b *

Compression is a little faster :)

v0.09b

Compression is a little better :)

v0.08b *

First release version of aPLib :)

v0.07b

Fixed a bug that gave errors when compressing multiple sets of data (thx x-otic!).

Cleaned up the code a little.

v0.06b

Changed the packer, so it uses a fixed amount of mem (about 1 meg).

A few bugs fixed.

v0.05b
First version of the aPLib library included.

v0.04b
Added the DJGPP fast asm unpacker.

v0.03b
Optimised the depacker a little.

v0.02b
Second try ;-P

v0.01b
First try!

Project started March 5th 1998.

Index