X1 User Manual

August 17, 1995

Contents

1 Requirements

CPU	386sx or better
RAM	a minimum of 4Mb memory is strongly recommended
OS	DOS 3.3 (or later) OS/2 (32-bit)

Get specific program versions for each OS.

Be aware that the memory reported by the archiver is the memory indirectly reported by DPMI - which is the memory actually "visible". Depending on the memory manager (DOS...OS/2...), disk-mapped memory could be used and the reported value may exceed the physically installed amount of memory. But it matters less for the program how memory is managed. It is more important, how much is allowed to be used (by the memory manager).

2 "What's this?"

This is an archiver. It compresses several files into a single file. It is entirely command-line driven. Compression and uncompression are combined into this single executable program. It has a speed-tightness-memory trade off that favors tightness and tries to make full use of the resources (memory) on the platform it is running on.

Most dominating features are:

- Introducing .x format archives
- Compatibility with ARJ, (PK)ZIP, LHA(RC), ZOO, HA
- Multi-lingual interface
- Contains "experimental" methods as a consequence of the exploration of hi-performance algorithms. "experimental" does not indicate instability - but rather that the method may not be supported backwards in every single future version.

The command-line interface is similar to that of ZOO and HA. But the option characters are more related to ARJ and LHA. The interface has little in common with PKZIP. With a little knowledge of common archivers, it should be possible to skip parts of this documentation.

3 "How-to-use" (command-line format and review of commands and options)

This is the specification for the program. If the program acts differently, then it is either caused by an error - or failure to properly implement the function according to this specification.

Using the description "archiver" for a program will produce a number of expectations - based on the abilities of other archivers. Consequently it becomes important to note not only what *has* - but also to some degree what *hasn't* actually been implemented.

If things mentioned in the text are not implemented, then a message "**Not implemented**" will precede the section. Further work should allow a reduction of the number of such messages.

3.1 Command-line format

```
X1 command[<options>..] <archive> [<filemask>..]
```

Definition:

`command[<options>..]` only one command allowed (first single character) followed by a sequence of options - with no spacing between command and options - or among options

`<archive>` is the archive name, with a default extension of .ZIP

`<filemask>` are the files to archive, defaulting to all files if none are specified. Wildcards *, ? can be used. Filenames are specified in normal extension of the path.

3.2 Command-line syntax rules

Command and option(s) can be specified in either upper or lower case.

Be aware that `x1` has two path exclude options instead of normally one. This is a consequence of rule 1 and 2.

1. (path included)

   By default, all paths mentioned on the command-line are added to the archive - unless exclusion option X is enabled.

2. (path included)

   By default, all paths encountered during recursive search (R option enabled) are added to the archive - unless exclusion option E is enabled. This is somewhat different from most other archivers.

3. (several filemasks relating to same dir)

   Filemask with no path specification assumes path of previous filemask.

   Explained in more detail:

   Assume x1 a test <arg[1]> .. <arg[i]> <arg[i+1]>... If <arg[i+1]> does not contain '\' ('/') or ':' then it is assumed to work under the same pathname as <arg_i> (which may work under the same pathname as <arg[i-1]>... etc.). What does this mean? It means that: x1 a test \dir1\dir2*.c *.h actually produces the same result as x1 a test \dir1\dir2*.c \dir1\dir2*.h. What does it not mean? Well, x1 a test dir1*.c dir2*.h does *not* work as x1 a test dir1*.c dir1\dir2*.h because dir2*.h contains \. Nor does x1 a test dir1 *.c work as x1 a test dir1*.* dir1*.c, which would actually be meaningless.

   The justification for this convention is entirely saved typing (notice how you did not need to repeat \dir1\dir2\ on the command line in the first example).

   Side-effects/drawbacks? If you want files from the current directory, you have to specify them all before listing individual directories... Still, this is just a convention, and should not restrict the user in any way.

4. (dir recorded in archive)

   Specifying a directory name on the command-line implies compression of all files in that directory. Notice that x1 a test \dir1\dir2 produces the same result as x1 a test \dir1\dir2\ or x1 a test \dir1\dir2*.*. And if recursive option is enabled, then all files higher in \dir1\dir2\.. are also processed.

   But watch out! If filenames are identical to pathnames, it may not always be obvious what the end-result will be...

3.2.1 Commands

- ? command (display help comments)

  This command is only used in the context x1 ?. It will display a brief review of the commands and options. The page shown is *not* the same page as shown when executing the program without parameters. If no parameters are specified, then only commands are listed (among them is "?").

If the information is displayed too fast, it is recommended to direct it into a file (`x1 ? >archive.lst`), which can subsequently be viewed by your favorite browser (Norton-Commander-F3) or in your favorite editor..

Many languages are supported. However, some languages were more difficult to understand than others. So it cannot be guaranteed that the meaning of the messages are always crystal clear. In case of doubt, then the english, danish, french, german, spanish pages should be consulted (in that priority order!).

(On PC) If your codepage settings are bad, then the result will be missing characters. So please use the little time to get a proper codepage setting.

- `A` command (add files to archive)

  Valid options: $,A,C,E,F,I,K,L,M,N,O,R,S,T,U,V,X,Y,Z

  This command has two modes of operation:

  - create archive and add files...
  - update (re-open) archive and add files...

  But basically it compresses and adds specified files to an archive. If the named archive does not exist, then one is created with the specified name. If it does exist, then it is updated. This means, that new files with the same pathnames as old archive entries will overwrite these old entries. A few exceptions exist (when compressing to solid modules) - but this is outside the scope right now.

  If a base directory is omitted, the default directory is assumed to be the current directory.

  If there are no files specified on the command-line, then all files (in the default directory) are added. This means that `x1 a test.x` compresses everything in the current directory.

  At no point is it possible to compress an archive being updated into the archive itself.

  If the extension of the target archive is not specified, then a default extension of ZIP is selected. If you wish to compress in another format, then it is necessary to specify the target archive with that format extension (ARJ, LZH, ...). It is not possible to compress to archives with completely arbitrary extensions.

  Filenames can be excluded from addition to the archive by use of the "Z" option.

  Example (compressing to standard format): `x1 a archive.x my_file.doc`
  Example (compressing to ZIP format): `x1 a archive.zip my_file.doc`

- `C` command (add comments to archive)

  Not implemented

  Valid options: I,C

  This command adds comments to an archive. Comments are possible in ARj, ZIP, ZOO, X - but not in LHA, HA formats. Reasons are differences in header structures.

- `D` command (delete files from archive)

  Valid options: I,S,Z

  In order to be deleted, the pathname entry in the archive has to match an argument on the command-line (upper- and lower-case is unimportant).

  If a filemask contains no path, then it takes effect on all entries in the archive, regardless of their paths. This means that `x1 d test.x *.pas` will even delete entries `dir1/dir2/*.pas` in the archive (if available). However, `x1 d test.x /dir1/dir2/*.pas` will only delete the latter type of files - not `*.pas` files with no pathname.

  If all files are requested to be deleted, then the entire archive is removed (deleted).

  Filenames can be excluded from deletion by use of the `Z` option.

  Example (deleting multiple entries): `x1 d archive.x *.pas *.doc`
  Example (deleting entire archive): `x1 d archive *.*`

- `E` command (extracting files without path)

  Valid options: $,A,F,I,N,O,T,W,Y,Z

  This command is identical to the `X` command with the `E` option. It is only provided because of the general consensus among archivers (to support this command).

  During extraction, the date/time stamps for the original files are restored together with the file contents. A crc check is performed in order to verify that the operation was successful. However, the file is verified before writing to disk, rather than by a read-verification check.

  Files can be extracted to a user specified directory on a user specified drive — but by default, files are extracted into the current directory. All extractions are done without paths (into a single directory).

  When extracting to a specific directory, the directory name should end with `/` or `\` as in the example `x1 e test.x c:\dir\`. Not terminating the directory name with such a character as in `x1 e test.x c:\dir` will cause the program to assume, that a file is attempted to be extracted from the archive.

  When extracting to a non-existing (command-line specified) directory, the user will be warned about the incident and consulted before further actions. The warning prompt presents several different solutions — from `(y)es` (create directory this time only) to `n(e)ver` (create at no time). The user answers by typing the letter in brackets. The actual words and letters can change for different language modules. It is possible to override the default question by activating one of the options `Y` or `N`.

  When extracting to a specified drive, it is allowed to let the drive specification end with `:` as illustrated by the example `x1 e test.x a:`. This is because no valid filename is assumed to end with that character. If several destination drives/directories are specified (as in example `x1 e test.x a: b: c:`) then the last specified drive/directory will be used for actual extraction (`c:`).

If trying to extract a file with the same name as an existing directory, then the extracted file will be skipped (unless smart overwriting is chosen with option `W`).

If trying to extract a file to a filename already used, the user will be warned about the incident and consulted before further actions. Only exception is 0 length files. They are not considered to contain any valuable information - and are overwritten without warnings. The warning prompt allows several different answers - from `(y)es` (overwrite this time only) to `n(e)ver` (overwrite at no time). The user answers by typing the letter in brackets. The actual words and letters can change for different language modules.

Example (extract file): `x1 e archive.x file_id.diz`

- `F` command (refresh files already in archive)

  Not implemented

  Valid options: E,F,I,X

  This command replaces entries in an archive with newer files, where possible. This means that the names and total number of files in the archive will remain constant. Only the contents of the archived files may change.

  Only files date/time stamps are checked, when searching for newer files. If files are older — or have same date/time stamps (but possible different size), then the files are skipped.

  Potential conflicts occur in the situation, where the archive contains entries with paths in their pathnames. Should it be necessary to specify these entries specificly? The answer chosen here is: no! Instead it is necessary to exclude them with the `E` option — if for some reason they are not to be refreshed as other files.

  Similar to the `A` command, if there are no files specified on the command-line, all files in the archive are attempted to be refreshed. This means that `x1 f test.x` attempts to refresh all files in `test.x` — while `x1 f test.x *.pas` only attempts to refresh `*.pas` files (including possibly `dir1/dir2/*.pas` files). Only `x1 fz test.x *.pas dir1/dir2/*.pas` makes sure that the latter files are excluded from the freshing process.

  Example (freshen all files): `x1 f test.x`

- `L` command (lists contents of the archive)

  Valid options: E,I,O,Z

  This command lists standard information about file entries in the archiver. This includes:

  - full pathname
  - compressed and uncompressed file size
  - compression ratio (% compressed/uncompressed)
  - date (in year-month-day format)
  - time (in hour-minute-second format)

- CRC (16/32 bit)
- DOS style file attributes
- compression method (identified by number rather than by name).

A character `*` appended at the end of the filename indicates that the file is encrypted and consequently cannot be extracted.

Dynamically "fitting" of filepath names is attempted. If there is room enough for large names, then they are listed on the same line as the additional info. Otherwise, the filename is put on a line for itself — and additional info is moved onto the next line.

It is possible to list all archives of specific formats (`x1 l *.lzh`) or all archives recognized by the archiver (`x1 l *`) — which only actually checks files with a recognizable extension (`.arj`, `.zip`, ...). It is possible to only check for specific files (`x1 l *.zip file_id.diz`) or multiple files (`x1 l *.zip *.nfo *.diz`).

If the information is displayed too fast, it is recommended to direct it into a file (`x1 l *.zip >archive.lst`), which can subsequently be viewed by your favorite browser (Norton-Commander-F3) or in your favorite editor..

Example (listing all `*.arj`, `*.zip`, ...): `x1 l *`

- `T` command (test integrity of archive contents)

  Valid options: E,F,I,O,Z

  This command really has two causes (modes of operation):

  - verify that the archiver have not made errors during compression
  - check for errors inflicted on the archive during transmission (storage media or hardware transmission channel faults)

  The first error is by far the most alarming. This is why it is also usually the error being tested for. Consequently the archiver is normally required to unpack the data (in memory).

  To ensure success of possible extraction, a validity function is provided in the shape of a CRC (cyclic-redundancy-checksum) check. Even though the process is not 100for most practical purposes.

  It is possible to test entire contents of several archives (`x1 t *.zip`), individual file occurrences in a group of archives (`x1 t *.zip *.doc`) or individual files in individual archives (`x1 t archive.x readme.doc`).

  If the archiver can be assumed to have done its job properly, then a simpler and faster CRC check can be made. This check is activated by using option `F`. How to know if the archiver has done its job properly? Well, try a traditional crc check once (immediately after compression). If the result is OK, then only subsequent transmission CRC errors can be expected.

  The user are informed about mismatches between the recorded CRC and the actually found CRC value. In the event of errors, it is very unlikely that data can be recovered — unless some form of ECC (error-correcting-code) has been activated. Updating files with CRC errors is often possible, but cannot be recommended.

If solid archives are involved, then CRC checks can take additional time. This is caused by the fact that extraction may presuppose the extraction of another file. In solid mode, data is compressed together in a longer stream, and this reduces accessibility.

Example (test all files): `x1 t test.x`

- `P` command (protect archive against errors)

  Valid options: I,L

  This command will cause error-correcting-code (ECC) to be appended to the archive — which should protect the archive to some extent from subsequent media or transmissions errors.

  Actually appending ECC more than once is prevented. Rather than using a simple methods more than once, it is recommended to use a more sophisticated method involving greater overhead.

  Just as there is a difference in the performance of compression methods on data — error correcting methods have to be optimized for specific types of errors. Unless this is done, the result will be increased overhead with low effect. Consequently more methods will be made available later.

  Default protection method used is "Level-1 ECC".

  Level-1 ECC: This type of ECC code is aimed at the casual user, who wants (low) safety with only a minimum of extra data overhead. The method is specialized to compensate for disk media errors. It can compensate for only one(!) lost sector. The sector size is configurable. Default is 512 bytes. If wanting to compensate for more than one sector of errors, higher levels of ECC protection should be used. If protecting a (N) byte long sector, then the additional data overhead will be (N+48) bytes. The 48 bytes covers a header including a crc for the header, a crc for the total file and test-vectors used to locate errors faster. The ECC data will be located in a continuous block at the beginning of the file.

  Level-2 ECC: Still awaits implementation.

  WARNING: Error-correcting methods only corrects a limited number of errors. A method will only correct errors according to its specification. Therefore: read the specification — and do not expect correction of errors beyond it!!! Error-correction only gives a limited security. It is consequently important not to have blind faith in the methods.

  Example: To protect against a single lost 512 byte sector (floppy disk), then the command used is `x1 p test.zip` or `x1 pm1 test.zip`. To add 1024 byte/sector ECC1 protection (which is 2x512 byte sectors, thus `m2`), then following command is used is `x1 pm2 test.zip`.

- `R` command (repair archive for possible errors)

  Valid options: I

  This command has two intentions (modes of operation):

  1. To actually repair errors in the archive

2. Only to verify that the archive is free from errors. A crc check only investigates compressed file entries in the archive — not possibly corrupted error-correcting-code (ECC). Obviously the "extended test" can (should) only be performed if there is actual ECC code to check.

Repairing archive with ECC protection: This command removes errors — either in the ECC data itself — or in the actual data, it should protect. If the errors are irrecoverable, then a warning will state this, and no further actions are taken. If the errors are recoverable, then they will be corrected, and a message will explain that the process was successful.

Repairing archive with no ECC protection: This has never been the intention of this command. In this archiver, there is no general "repair" function for damaged archives. Instead the user is urged to attempt extraction of the archive — and re-archive the data again. The problems with providing a general "repair" function are well known from the program `PKZipFix` provided with the archiver `PKZIP`. The repair program just updates the archive header structure - without checking the integrity of individual files. Files can then be appended to the archive. However, without an integrity test, there is no way of knowing if the archive contains additional errors. Not checking for this possibility, may cause a false feeling of security. It appears much better to only allow extraction from the archive - allowing the user to directly verify the number of errors. If archives are stored within archives, then there exist dangerous possibilities for mistakes. In such cases, it is best that the user tries to evaluate the situation. If somehow people would be interested (and willing to pay!), then an advanced error-recovery program may be constructed for some/all of the many different formats. But it would not change the fact, that most errors normally are irrecoverable.

Example: `x1 r test.x`

- `U` command (unprotect archive — removing error-correcting-code)

  Valid options: I

  This command retraces the action taken by the `P` command. Removal will be attempted on any available error-correcting-code (ECC). Failure to locate ECC code may be caused by two reasons: no ECC code there in the first place — or errors blocking the detection of the ECC code (header). In the latter case it should be recommended to attempt removal of errors.

  It is recommended to check for possible errors before removing the ECC code. Even though the process may manage to locate the ECC code — then the archive can still contain errors at other positions in the file. No checks for errors are done when actually removing ECC code (in order to speed up the process).

- `X` command (extracting files with full pathnames)

  Valid options: $,A,E,F,I,N,O,T,W,Y,Z

  Main difference from the `E` command is that files are extracted with pathnames (as recorded in archive). The pathnames can be viewed before extraction with the `L` command.

Files can be extracted to a user specified directory on a user specified drive — but by default, files are extracted into the current directory, and possible paths are built from the current directory. Some paths cannot be used as extension of the current directory. That is paths which contain drive names or specify specific root directories. Such paths will be positioned on the disk according to their absolute specifications.

When extracting to a specified directory or drive, paths will be built from the specified position. For example `x1 x test.x a:` will try to extract all files to `a:` (while building paths from the current directory on that drive).

When file extraction requires a non-existing path on the disk, the situation is treated as when extracting to a non-existing (command-line specifed) directory. The user will be warned about the incident and consulted before further actions.

3.2.2 Options

- `\$` option (add/extract disk volume label)

  Not implemented

  Compression: Adds the disk label (for the first specified filemask) to the archive. In the example `x1 a text.x *.* a:*.* b:*.* c:*.*` the disk label for the current drive is added to the archive — because default drive was used in the first filemask. `x1 a text.x a:*.* b:*.* c:*.*` causes recording of disk label for drive `a:`.

  Extraction: Restores disk label (if present) from the archive to the disk, where files are extracted. Because only one disk label is allowed in the archive, no confuson should be possible.

- `A` option (attributes store/restore)

  Not implemented

  General note: this option was mainly intended for “special attributes” such as extended attributes in OS/2. Simple attributes as those found in DOS are always restored.

  Compression: Then the function is to save all attributes (possibly extended) in the archive.

  Extraction: Then the function is to restore the (possibly extended) attributes. Operating system (OS) dependant attributes can only be restored under the same OS. In case of OS conflict, the attributes are either not restored — or a suitable (simple) replacement is chosen.

- `C` option (fetch comment from a file)

  Not implemented

  Used with `C` command only. There are two ways to add comments to an archive:

  - type it in from the keyboard
  - insert characters from a file

If the comment is to be typed in by keyboard, then option `C` is not activated. If comments come from a file, then the option should be used.

- `E` option (exclude path from filenames)

  Compression: The function is to exclude paths found during a possible recursive search for files. Unless activated, all paths are included in the archiver by default.

  Extraction: The function is to make sure that all files are extracted into the same directory - regardless of the paths recorded in the archive.

  File refreshing: The function is to avoid refreshing files with paths in the archive. Otherwise, all files are refreshed regardless of path by default.

- `F` option (force file movement in/out of archive)

  Not implemented

  Compression: The function is to delete files from disk as soon as they have all been compressed and the entire operation looks successful. This is to ensure extra safety — so if the operation is prematurely aborted, then it should not result in lost data.

  Extraction: Then the function is to reduce the size of the archive — by deleting the entries of the extracted files.

- `I` option (quiet mode)

  Not implemented

  This option suppresses all output to the screen. This include the standard header and pieces of software to shell out and let the archiver process data — without having the screen messed up with irrelevant symbols.

- `K` option (overwrite existing archive)

  Not implemented

  Compression: Then the function is to cause overwriting of a possibly already existing archive. Otherwise the archive would have been updated — and new entries added to existing.

- `L<number>` option (level of compression)

  Compression: Then option specified a more detailed level of compression. This may mean use of more memory — or degraded/faster compression — all depending on the selected compression method.

  .x method 4: then valid levels are 0,1,2,3. Each level determines a different memory model.

  other methods: no effect

- `M<method>` option (method of compression)

  Compression: Specifies the compression method to be used. Method is specified as a number 0..256. If the number cannot be associated with a valid compression method, then an error message is displayed.

- `N` option ("NO" assumed at all enquiries)

  Extraction: Option `N` is complementary to option `Y`. Option `N` causes all potential questions to be answered with a "NO". This will avoid any attempts to halt the process and ask the user for advice. It mainly affects possible overwriting of files, creation of directories etc.

- `O` option (order files according to name, size, date, etc.)

  Not implemented

  There should always be two 'O' characters in the option string on the command-line. Between the `O...O` will be sort sub-options — which defines the sort keys — and their priority. However, if no other options is to succeed the 'O' option then the last 'O' can be left out.

  Supported sort sub-options are:

  - N: name
  - S: size
  - D: date
  - T: time
  - (b: best compression)

  So for example, `OSNO` mean: first sort according to size — and if two files have same size, then sort according to name.

  Compression: If compressing to a new archive, then files are inserted into the archive in the order specified by the sorting keys. If updating an archive, then existing entries will be updated in their current order — but new entries will be sorted according to the options.

  Extraction: Then files are extracted in the order specified by the options.

  Example: Sorting compressed files according to (1) date (2) size and (3) name: `x1 aodsno test.x *.*`

- `R` option (recursive search through subdirectories)

  Compression: Without this option, only the files in the command-line specified directories will be processed. Where no directories are explicitly defined, here the current directory is assumed. But activating the option will result in additional processing of all files in subsequent subdirectories. Subdirectories are visited recursively in the order they are found on the disk. The directory paths recorded in the archive are extended according to the subdirectories visited — unless option `E` is activated. Files are visited twice: first during an initial scan — and later when actually processing the file as an archive entry. The initial scan is performed in order to allow an overall progress-indicator — and allow size-estimation for solid headers (with several file entries).

- `S` option (change archive date/time to newest entry in archive)

  Not implemented

  Compression: Three practical possibilities seem possible when setting the date/time mark according to the newest file:

  - selecting the newest date/time among the entries added to an updated archive
  - selecting the newest date/time among the entries already present in an archive — before starting to update it
  - select the newest date/time among all files after all updating is done.

  But in the current implementation, only the last approach has been implemented.

  An easy way to set the date/time tag, when not wishing to add extra files is by `x1 lis test.x`, which causes listing (but not displaying) of the contents - and setting the date/time tag in the process.

- `T` option (set file(s) date/time tag according to current time)

  Not implemented

  Compression: The option sets date/time tag for files added to the archive to the current date/time. The original date/time for the files are ignored.

  Extraction: Brands the extracted file with the current time — rather than the time recorded in the archiver.

  Note: all files in one process will receive the same date/time tag. Even though files may be processed with a few seconds difference, then they will all receive the time, when the process was started.

- `U` option (solid compression mode)

  Compression: If only one file is to be compressed, then the option will have no effect. But if more than one file are compressed — then they are compressed in "solid mode". This means, that files are compressed as a continuous data stream (as if they were concatenated) — which for related files should result in increased compression tightness.

  For optimal compression, files should be grouped intelligently, so that most related files are placed closest together. However, this is not automaticly attempted in the current implementation.

- `V<size>` option (multivolume processing)

  Not implemented

  Compression: If the *size* is not specified, then compression will be done to a (multi-volume) archive limited in upper size by the currently available disk space. After each volume, a promt will appear (unless option `Y` activated) — which in case of a removable disk drive will allow insertion of a new disk. When allowed, the compression will continue with another archive part. If *size* is specified, then compression is done to a (multi-volume) archive limited in upper size by the specified value. In any case, the end result should be a sequence of archives with identical names - and changing extentions "x00", "x01",... where "x00" contains the first data compressed. It is not possible to update a multi-volume archive in multi-volume mode. But it is possible to update a multi-volume archive with additional ordinary entries - or update a non-volume archive with multi-volume entries.

Extraction: Here *size* has no relevance. Extraction should only be attempted on a complete sequence of multi-volume archives. Attempting to extract an individual multi-volume part is possible - but will only be successfull for non-continous file entries, not spanning over more than one volume. Prompting for each volume part is done unless option `Y` is activated.

- `W` option (overwrite extracted smartly)

  Not implemented

  Extraction: This option attempts to intelligently resolve situations, where extracted files risk overwriting already existing files. Traditionally the situations would be resolved by asking the user if overwriting was allowed — and avoiding extraction if not. But a third possibility is possible by extracting the files to slightly changed filenames. Filename extensions 00X, 01X, ... (not to be confused with actual .x type archives) are used for collision-resolved extraction.

- `X` option (exclude path in filenames on command-line)

  Compression: By default, all paths written on the command-line will be included in pathnames (for files added to the archive). If a directory is specified with `x1 a test.x dir\*.*` then pathnames `dir\file1`, `dir\file2`, ... are recorded in the archive. Option `X` avoids the default recording of paths on the command-line. `x1 ax test.x dir\*.*` will record previous file entries as `file1`, `file2`, ... Please notice, that the option does not affect pathnames recorded during recursive search on a disk (option `R`). To eliminate paths in those cases, option `E` is required.

  Extraction: Then the option will have no effect — only option `E` should be used to exclude full paths from files to be extracted.

- `Y` option ("YES" assumed at all enquiries)

  Extraction: Option `Y` is complementary to option `N`. Option `Y` causes all potential questions to be answered with a "YES". This will avoid any attempts to halt the process and ask the user for advice. It mainly affects possible overwriting of files, creation of directories etc.

- `Z<number>` option (exclude files otherwise archived)

  Compression: When the option is not used, all filemasks on the command-line specifie files to be included in the archive. When the option is used, but not succeeded by a number (as specified), then that number defaults to 1. The number specifies how many of the last filemasks on the command-line specifieexcluded files. This convention does not allow mixing of include-masks and exclude-masks. All(!) exclude-masks have to be specified together — and at the end of the command-line. In the example `x1 az3 test.x test1.* test2.* *.bak *.swp *.tmp` the number 3 follows option `Z` and tells that the last 3 filemasks `*.bak *.swp *.tmp` mark files, which should not(!) be added to the archive. The number counts backward from the end of the command-line and are consequently not affected by the number of include-masks.

In case the exclude option number equals or exceeds the total number of filemasks on the command-line, then by default the files `*.*` are assumed to be included — and all specified files excluded.

4 Environment variables

At present, only one environment variable is supported. This is `COUNTRY`, which is used for language/character-set selection.

In DOS, the environment variable COUNTRY can be used to specify the currently active country and character set. Special language-dependant characters are supported in the upper ASCII character set (offsets 128 to 255) — and may change with the configuration. Depending on character set, the language-dependant characters may occur at different offsets in the extended ASCII table — if available at all. When needing such characters, it becomes important first to identify the character set.

For example, the configuration for an US keyboard could be:

```
COUNTRY=001,437
```

The `001` specifies US language, while `437` specifies codepage 437 (US character set). Consult you DOS manuals for a detailed list of numbers and association. For `x1`, `001` should cause messages to be displayed in english. The `437` causes possible extended characters to be located at offsets corresponding to codepage 437.

If for some reason the environment variable COUNTRY is preferred not used — then the other variable LANGUAGE can be used instead. Same syntax is required.

Notice, that if for some reason, a sentence is not supported in the current language, then the original english message will be displayed (in lack of a better choice).

Other environment variables are not supported at this current point.

5 Formats and compression methods (user level)

First an overview of the header formats and methods supported.

`x1` archiver as of version 0.94a ("+" indicates support and "–" indicates lack of support):

5.1 About the ARJ module

An example in compressing to the ARJ header format could be:

```
x1 a test.arj *.*
```

This will cause all files in the current directory to be compressed to the archive `test.arj` (by default with method 1). Store files with: `x1 am0 test.arj *.*` Please notice that ARJ multivolumes are not (yet) supported.

archiver name	capable of listing	capable of unpacking	capable of packing	compressing methods supported	uncompressing methods supported	comments
arj	+	+	+	0,1	0,1,2,3,4	1,..,4 are basicly identical
zip	+	+	+	0,1,8	0,1,7,8	1:shrink,7:implode,8:deflate
lzh/lha	+	+	+	0,5	0,1,5	no AMIGA–LHA support (no Amiga!)
zoo	+	+	+	0,2	0,2	
ha	+	+	+	0,1,2	0,1,2	
tar	+	–	–	–	–	
tgz	+	+	+	8	8	requires gzip'ed TAR headers
rar	+	–	(+)	0	0	
arc	+	–	–	–	–	
pak	+	–	–	–	–	
sqz	+	–	(+)	0	0	
arx	+	–	(+)	0	0,1 (5)	algorithms as LHA1.13
put	+	+	+	0,5	0,5	algorithms as LHA2.x
x	+	+	+	0,1,..	0,1,..	

(+) means that storing only is not much fun... :-)

5.2 About the ZIP module

An example in compressing to the ZIP header format could be:

```
x1 a test.zip *.*
```

This will cause all files to be compressed — by default with the ZIP "deflate" method known from ZIP versions 2.x. If backward compatibility is an issue, then it is also possible to compress with an older method:

```
x1 am1 test.zip *.*
```

This applies the method "shrink" from ZIP versions 0.9 and 1.1. This old method is not intended for any serious use. It dates back to my first humble experiments with the ZIP format in 1991.... Testing archive integrity (after compression) is done with:

```
x1 t test.x
```

or

```
x1 t test.x *.*
```

5.3 About the LHA module

An example in compressing to the LHA header format could be:

```
x1 a test.lha *.*
```

or

`x1 a test.lzh *.*`

The extensions `.lha` and `.lzh` are supported on equal terms (both caught if listing `x1 l *`). Method 5 is default.

5.4 About the ZOO module

An example in compressing to the ZOO header format could be:

`x1 a test.zoo *.*`

WARNING: the zoo format specifies two header types and only one is currently supported by `x1` (the all-dominating type 2). This may cause problems, when `x1` updates old zoo-created archives (header type 1?). Maybe I am too cautious — cannot tell.... E-mail me some examples with old headers!!!

5.5 About the HA module

An example in compressing to the HA header format could be:

`x1 a test.ha *.*`

This causes all files in current directory to be archived (by default with method 1..ASC). If requesting tighter compression by a 4th order PPM, then it would be more appropriate to use (HSC):

`x1 am2 test.ha *.*`

Letting the computer try both methods and automaticly choose the best is done with: `x1 am# test.ha *.*`

5.6 About the TGZ module

An example in compressing to the TAR+GZIP header format could be:

`x1 a test.tgz *.*`

This causes all files to be wrapped in TAR headers and consequently compressed into GZIP format. On platforms supporting longer filenames, the more appropriate filename extension should be `.tar.gz`. When listing the contents of such a file, total decompression (in memory) is necessary - so a little extra patience may be required.

`x1 l test.x`

Currently, it is not possible to extract the contents of a GZIP file as a single file.

5.7 About the X module

A thing that not directly relates to any compression method — but differentiatess the X header format from most other archives — is the double-crc recording (ability of fast CRC check). CRC is calculated for both uncompressed and compressed data. When just wanting to check for transmission errors, checking the latter CRC field will be sufficient.

Because the size of the compressed data are usually smaller than the original size, the work involved with the extra crc-calculation are usually insignificant. It does not slow down any process in a significant way.

Furthermore, notice that .x archives may contain more than one file with exactly the same pathname. This cannot be avoided when solid entries are involved. If the solid entry with the duplicate filename(s) is deleted — then other files are deleted (compressed together) — which will result in lost data. This problem could be avoided by "pretending" that a duplicate filename just ain't there (not showing it, as in UC2 or HPACK). But it seemed reasonable, that when it is actually there, the user should see it... A convention!

Method number	Name	Packing example	Comments
1	XLZ	`x1 am1 test.x *.*`	64k window LZ77
2	XSC	`x1 am2 test.x *.*`	requires DOS mem. (640k)
3	XXSC	`x1 am3 test.x *.*`	
4	XXC	`x1 am4 test.x *.*`	Variable memory selection possible
5	LZP	`x1 am5 test.x *.*`	
best method	-	`x1 am# test.x *.*`	

NOTE: best method will first try method 3, then 1. The method finally accepted will be the one offering the tightest result.

WARNING: method 4 is currently target for additional work... Expect no backward/forward compatibility here!!!!

The X module supports solid format. An example while compressing in solid format could be: `x1 aum1 test.x *.*` An archive can contain several independent solid entries — and solid entries can be freely mixed with normal single-file entries. Once created, a solid entry cannot be updated.

To verify compressed file by integrity-checking ability to uncompress:

```
x1 t test.x *.*
```

To verify compressed file by checking for transmission errors:

```
x1 tf test.x *.*
```

Yes, this gives the `F` option ("force" file in/out) another meaning ("faster" CRC check), when validating files. But this does not restrict the use of the option in the original meaning, because it would make no sense to "force" files out of an archive during a test process.

Attempting to "fast"-CRC check other formats, which do not support double CRC will just result in a normal CRC check.

A brief word on method 4: It works with the "compression level" option `l` in the way that:

```
x1 am4l2 test.x *.*
```

... actually means compression on level 2 (<4Mb memory).

Level	Memory	Comment
0	640kb	memory manager may require min. 1Mb
1	3Mb	-
2	7Mb	-
3	16Mb	could not test it - no 16Mb RAM

Be warned, that the method may change again!

6 Temporary file names

While compressing, the program uses a temporary file `X???????.SWP` (where ??..?? are randomly chosen). It contains the new archive being built. If the entire operation seems successful, the file is renamed to the requested name.

While decompressing, an extracted archive entry may be requested to overwrite the archive itself. In that case, the original archive is temporarily renamed to a neutral name (`X???????.SWP`) — and deleted, once all requested entries are extracted.

At no point is more than 3 files opened (when updating an old archive). Once started, the program will use maximum one temporary file — and only one name for it (`X???????.SWP`).

The extension may appear a bit misguiding. The temporary file is not a swapfile. The file is renamed at the end — contents is not copied elsewhere. But the filename extension should give associations to other programs (Borland compilers..), which may cause less hesitations, when the user considers deleting a file.

Because swap files are not really necessary, no special work directories are supported. For optimal speed, the compression process should take place on a virtual (RAM) drive — and the resulting file(s) subsequently copied to the desired destination.

7 Exit return codes

Internal reference	Value	Comment
EXIT_OK	0	No error
EXIT_INT_ERR	1	Internal error (?)
EXIT_NO_MEM	2	Out of memory
EXIT_NO_DISK	3	Out of disk space
EXIT_NO_OPEN	4	Cannot open archive file
EXIT_NO_TMP	5	Cannot open temporary file
EXIT_NO_FOUND	6	Cannot find matching archive(s)
EXIT_NO_PATH	7	Cannot find path
EXIT_NO_BASE	8	Cannot access base directory
EXIT_NO_MKDIR	9	Cannot create directory
EXIT_BREAK	10	User interrupt
EXIT_FILE_ERR	11	Unknown file error - cannot read from file
EXIT_BAD_HDR	12	Archive directory (partly) corrupted
EXIT_BAD_EXT	13	Contents of archive does not correspond to extension
EXIT_BAD_ARC	14	An archive at all?
EXIT_LONG_NO	15	Unsupported long arg.format - or too many arg.
EXIT_LONG_PATH	16	Path too long
EXIT_NO_OVERRIDE	17	Cannot override base path
EXIT_NEST	18	Dir's nested too deeply
EXIT_SCRIPT_ERR	19	Error(s) in script file
EXIT_NOTHING	20	Nothing to do
EXIT_UNSP	21	Method not supported in this version..
EXIT_COM	22	Unknown command
EXIT_OPT	23	Unknown option
EXIT_OPT_DIR	24	Unknown option
EXIT_OPT_WR	25	Unknown option
EXIT_OPT_VIEW	26	Unknown option
EXIT_WC_FORM	27	Wildcard inappropriate use
EXIT_WC_NAME	28	Wildcard inappropriate use
EXIT_WC_COMPL	29	Wildcard inappropriate use
EXIT_CHG_WRPROT	30	Bad attempt to change (write protected) archive
EXIT_CHG_DEL	31	Bad attempt to change (deleted) archive
EXIT_CHG_UNI	32	Bad attempt to change (block compress) archive
EXIT_CHG_MULTI	33	Bad attempt to change (multivol) archive
EXIT_CHG_CR	34	Bad attempt to change (encrypted) archive
EXIT_CHG_UNCR	35	Bad attempt to change (unencrypted) archive
EXIT_BAD_KEYFILE	36	Bad keyfile (cryptation)
EXIT_PASS_NO	37	no valid (user)ID (cryptation)
EXIT_PASS_DIFF	38	Password verification failed (cryptation)
EXIT_SECURITY	39	General secur/encryp error (cryptation)
EXIT_NOCRYPT	40	Cannot handle encryp.archive (cryptation)