# XpkMaster

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>XpkMaster | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | August 6, 2022 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# XpkMaster

## 1.1 Welcome to the XPK distribution

```
                Welcome to the eXternal PacKer (XPK) library system for easier ↩
                   handling
of crunching and decrunching.


                About
                      About this distribution.

                Archive Contents
                    Contents of distributed archives.

                Version Info
                      What changes are made.


                GNU - License
                    License for some of added libs/programs.

                Philosophy
                      Some background information.

                Preferences
                      Preferences system of version 4.


                SubLibs
                      Documentation for supplied sublibraries.

                C-Utils
                      Documentation of the included programs.

                XPK programs
                      XPK supporting/using programs.


                Contacts
                      Addresses of people related to XPK.
```

```
Try WWW addresses
  http://www.amigaworld.com/support/xpkmaster/
or
  http://rcswww.urz.tu-dresden.de/~stoecker/xpkmaster.html

Here all files are accessable (also the xpk_Crypt.lha archive).
```

## 1.2 The contents of the distribution

```
                xpk_User.lha (util/pack)
  C/        Various
                programs
                 using XPK.
  catalogs/     Catalog files for use with locale.library.
  EnvArc      Preferences file example.
  Libs/xpkmaster.library  The heart of the XPK system.
  Libs/compressors/   Compression
                sublibraries
                .
  Libs_1.3/     xpkmaster.library for OS 1.3.
  Libs_68020+/      68020+ versions of some
                sublibraries
                .
  Prefs/      Preferences program.
  Install     Installer script for complete distribution.
  XpkMaster.guide   This documentation.

xpk_Crypt.lha (not distributed in USA, in Germany util/crypt)
  libs/compressors/   Encryption
                sublibraries
                .
  source/     Source files of encryption libs.
 This archive can be accessed only on German Aminet servers or by calling
 my WWW page (See below).

xpk_Develop.lha (util/pack)
  Autodocs/     Programmer documentations of libraries.
  HotHelp/      HotHelp documentation.
  Include/      Includes for programmers.

xpk_Source.lha (util/pack)
        Sources to libraries and utility programs.
```

## 1.3 xpk programs

```
                XpkArchive System: (2.0)
------------------------
As stated in
                Philosophy
                 above the xpkmaster.library there exists
also the xpkarchive.library, which handles archive creation like LhA or Zoo.
Please have a look at this package too!
```

```
Author:
                Matthias Meixner
                Where to find:  in Aminet as util/arc/XpkArchivePackage??.lha

XFH: (1.40)
-----------
XFH-Handler is a DOS handler which uses xpkmaster.library to provide
transparent access to compressed files in a given directory or partition.
All compression/decompression is done automatically by the handler when
files are read or written. Compression is optional and may be switched at
any time, allowing for fine control over storage of data. The compression
method may be changed at will. Decompression is always automatic, you
do not have to care about which compressor was used to create the files.

Author:
                Matthias Scheler
                Where to find:  in Aminet as util/pack/XFH.lha

XPKatana: (1.2)
---------------
Full-featured GUI for XPK, allows (un)packing, features a complete ARexx
port, supports batch jobs, etc... Also supports FileID.library for
filetype identification, and xfdmaster.library for alien packer
decrunching, making it possible to repack most alien packing formats into
a regular XPK format.

Author:   Eric Sauvageau <sauvae00@libertel.montreal.qc.ca>
Where to find:  in Aminet as util/pack/xpkatana??.lha

XPK-KNIGHT: (1.05)
------------------
XPK-KNIGHT is a free configurable, easy graphic user interface for
comfortable usage of most of the important functions whithin the
xpkmaster.library and its correspondending XPK packers.

Author:   Alexander Grossberger <nobody@betei.franken.de>
Where to find:  in Aminet as util/pack/xpk-Knight_???.lha

XpkCybPrefs: (1.0)
------------------
XpkCybPrefs is an alternative to XpkMasterPrefs. It provides a transparent
"intelligent" packing, according to the filetype (by simply setting "USER"
as xpk packer and configuring once for all, you get everything working).
Some features: Size conditions, "Multipacking" (trying multiple packers on
data, keeping the best or fastest-decrunching etc... one!!), report-window
with full info, user-choice requester ETC ETC... (100% user-configurable)

Author:   Alexis Nasr <nasr@hol.fr>
Where to find:  in Aminet as util/pack/XpkCybPrefs.lha
```

## 1.4 about

```
                This distribution was created by
                Dirk Stöcker
```

```
                 in October 1996.

The concept, the most stuff and docs were made by
                Bryan Ford
                 and
                Urban

                Dominik Müller
                . A lot of bug fixes made
                Christian von Roques
                . I code xpk
now, because the above mentioned person have no time to do further support.

I made the distribution up of the newest data I could get, but there may
still be some older parts. Please tell me what is wrong or to old and send
me newer stuff.

To the authors of the sublibraries:

I changed those libraries, I had the source for, a little bit. This means:
  - added a nice library header giving the right version information
  - remove some of the now obsolete version information
  - added 1 to the revision number and set the actual date
  - I did not change anything concerning the work of the library
Tell me, if this is not OK! In this case also send me the newest version of
your lib too. In the developers dir I included a nearly standard xpksublib
header, so it should also be possible for you, to add correct version
information.

Also if it is OK and you have a newer version, please send me for including
in the distribution.

To the authors of programs supporting XPK:

Please send me a short description (10 lines maximum) of your program. I
compile a list of XPK supporting programs from this information.
Please include information, where to find your program (e.g. in Aminet
util/pack).

About this documentation:

I created this documentation starting from a lot of single files. To cut
down the size a little bit down I had to omit much useless or double
information. If I deleted something I should not, please tell me and I will
reincorporate it. Error reports and newer docs or corrected contact
addresses are welcome too. Most of them are probably outdated.

If someone feels, like translating the Installer script or the catalog
files for xpkmaster.library and XpkMaster prefs program, send it, and I
will include the stuff in next release.
```

## 1.5 history

```
                When the version number has a character attached, the xpkmaster ↩
                   library
```

did not change, but only the distribution was updated.

Dirk Stöcker's changes:

4.14a: fixed Installer script, rewrote xDir, added more language files

4.14: fixed low-mem-bug in NUKE and DUKE, little changes in master lib, fixed IDEA bug (thanks to Alexis Nasr for finding a solution), fixed bug in XpkMasterPrefs, added some parts to the docs, added length check and OR specifier to file pattern check, added pattern matching for xBench

4.13: fixed Installer script, added xBench ALL and SAVE option, removed error in Expunge code of library, fixed xQuery

4.1 - 4.12: added automatic password request and preferences system, cleared the code a bit and removed obsolete code, removed xDrop from distribution, wrote new xBench utility.

4.0: added 5 new functions, fixed some little bugs, added first stage of preferences system (XpkMaster and XpkMasterPrefs programs), library is now OS2.0+ only

3.12: OS1.3 version, removed locale and utility library stuff

3.11: recompiled the library with SAS 6.57, cleaned up the complete package, removed XFH

3.10: I reworked the source and added locale.library support, a better library header and another way of debuging.
NOTE: The include files moved to xpk dir. Before it was libraries.

Changes made by Christian von Roques :

3.9: The file-opening routine did not recognize, that files shorter than 4 bytes have a length at all. It now assumes, that if it cannot read 4 bytes, the number of bytes it was able to read were the whole file. This hopefully puts an end to the notorious series of bugs with files shorter than 4 bytes.

3.8: Fixed zero-padding while calculating the block-checksum to not munge innocent memory behind the input-buffer, if the input is non-compressible. This bug was present since V2.x, but has been hidden by the stupidity I removed in V3.2.

3.7: The length of the uncompressed data is computed and stored in the global header of xpkfiles, even if they were written using XpkWrite.

3.6: I broke something in V3.4 [one part depended of anotherones stupid behavior, and I made this part behave less stupid, which broke the other. --- What a marvelous design :-(] Compressing Mem->Mem should work again.

3.5: -Added some debug-messages to XpkClose. If xpkmaster.library returns an error, please report both the error-code as well as the error message.
     -fixed a stupid typo in the to-memory hook-function. Compressing to an

```
outbuf should not give bogus error-codes anymore.
     -Speedup checksum computations using Duffs device.

3.4: Fixed a longstanding memory thrashing bug which first surfaced in
version 3.3. Twiddled with the handling of files shorten than 4 bytes while
still being compatible to xpkmaster.library V2.5, but it still did not
work.

3.3: Files shorten than 4 bytes were not planed for at all. It will not
crash anymore, but still does not handle them correctly.

3.2: Seek()ing in memory and files has been rewritten from the ground.

3.1: Both memory and file IO has radically been cleaned up.

3.0: Cleaned up the source of xpkmaster.library some bit, but it still
is a mess. XpkSubParams now have a new field LibVersion containing
the major version number of the sublibrary used to XpksPack the
chunk. This can be used to create new backwards compatible versions of
sublibraries. Take a look at the source of V2.x of
                xpkFAST.library
                for an example of this. [note: V2.x of
                FAST
                 was not sufficiently better
than V1.x to warrant a new release.]
```

## 1.6 Preferences system overview

```
                The preferences system allows packing of files depending on their ↩
                   file type,
their size or name. It is useful especially for programs handling a lot of
different files like backup tools. With type depending packing you get a
faster and shorter backup. But the system may be useful for lots of other
situations.

It consists of 2 parts:

1) Program for preferences settings in "Prefs" directory: XpkMaster
2) Shell utility to handle preferences files on run-time: XpkMasterPrefs

* Programmers should read the autodoc file xpkpreferences.doc. There is
* told how preferences are handled and how you can replace the preferences
* system with you own one. There already exists another system called
* XpkCybPrefs.

The packer preference settings are used, when selecting USER packer type.
Global preferences are used all time when needed. NOTE: some programs may
forbid use of preferences.

1) XpkMaster
------------

It is a normal preferences program like Serial, Printer, Locale, ...

This program is used to set global xpkmaster defines (TimeOut, use of xex
```

libraries, ...) and to define file types and corresponding packing methods.

The screen is divided into left and right part. On left side there is a list of all filetypes. 4 gadgets allow manipulation of the entries: move up and down type, delete type and add new type. First entry always exists and cannot be deleted/moved. This is default entry. It’s name can be changed, but after reloading saved data the default name applies again. When adding a new type, the data from default tye is copied into new type.

The right side allows manipulation of the filetype:

Name Pattern:
Specify a standard AmigaDOS pattern describing the filetype. Using field File Pattern is better, because some programs do not submit file names or use dummy names. It is always better to find a File Pattern description, than a Name Pattern.

File Pattern:
This allows to describe a file type by comparing the file contents with a defined pattern. See
filepatterns
for pattern description and examples.

Library Name:
Four letter ID of library you want use for packing.

Mode:
Mode used for packing. This is a decimal number from 0 to 100.

ChunkSize:
ChunkSize used for packing. See autodoc files for better description. Most time it should be 0.

PackMode:
Define if you want to pack the file, return an error or do not pack the file. When selected ’return error’, the caller program gets told, that packing was not possible. When selecting ’do not pack’, the file is passed through without changes. If these two type are selected, Library name, Mode and ChunkSize fields are disabled.

Now the last part, the global defines. These defines are not file type specific, but xpkmaster global.

Use XFD:
Should xpkmaster use xfdmaster.library for decrunching custom file types, or not? This is not implemented at the moment.

Use XEX:
Should xpkmaster install list, where the scan routines of xex libraries are hold. This is necessary to recognize custom filetypes on decrunching. It is not imlemented at the moment.

Auto Password:
Should xpkmaster use automatic password requester, when password is needed, but not supplied. Password requester quits after a definable time value.

Timeout:

```
Define the time password request stays open with no user interaction. After
that time the requester automatically quits, when no user-action happend.
Default is 120 seconds.

The last three gadgets (Save, Use, Cancel) and the menus are equal to
standard prefs files and easy understandable.

The patterns: When both pattern types are specified, both must be true,
to specify a filetype.

Example: GIF files
1) filepattern and no namepattern is specified:
  XpkMaster recognizes all GIF files
2) filepattern and name pattern (#?.gif) is specified:
  XpkMaster recognizes all GIF files, ending with .gif
3) on name pattern is specified:
  XpkMaster recognizes all files ending with #?.gif (also if these files
  are not GIF files!)

A correct file pattern should most times be enough and better as a name
pattern. So example 1 is nearly in all cases the best way!

2) XpkMasterPrefs
-----------------

This program is like IPrefs. It loads the file ENV:xpkmaster.prefs and
makes it possible for xpkmaster.library to access the data. How this is
done is explained in the autodoc file xpkpreferences.doc. Everytime you
change the file ENV:xpkmaster.prefs by selecting USE or SAVE in
XpkMaster preferences program, the data is updated automatically. So after
selecting USE, xpkmaster.library uses the new settings.

NOTE: The filetypes are scanned in same order, as you entered them in
XpkMaster preferences program. There is only one exception: The default
type (displayed first always) is not scanned, but used when no other type
matches (which means it is the last always!).

How to install:
Copy it to C: directory and start the program before you use xpkmaster
library first. The best place is S:User-StartUp (or S:StartUp-Sequence in
some cases). There you may add the line

  Run >NIL: XpkMasterPrefs

The rest will be done automatically by the program. To kill the program
call it with "XpkMasterPrefs QUIT" or send it a CTRL-C break signal.
```

## 1.7 filepatterns

```
The file pattern consists of a mode describing lower case character
(l, h, m, v, r, g, s) and corresponding hexadecimal data. It helps to
describes file type specific parts.
You may specify different types in one string by seperating the types by
'|'. So text1|text2 means that the file has to match either text1 or text2.
```

```
Control Characters:

For these control caracters the data consists of a $/hex value with upto
8 digits (ULONG) [option m allows only 4 digits (UWORD)].

l - filesize must be shorter (LOWER) than specified value
h - filesize must be larger (HIGHER) than specified value
  When using these options, be sure you do not forget any size. When
  specifying a type l100 and one type h100, the size 100 is missing!

m - Move to position in the file. The first byte has position 0.
  Note: The real maximum move-size depends on the buffer value set in
  XpkMasterPrefs program. Currently this match part are the first 2048
  bytes.

The following numbers/strings are specified using $/hex values and must be
given as bytes (2 digits). You may repeat data behind these without
adding the command each time (f.e. v40v41 is equal to v4041).

v - Match specified BYTES at the current position in the buffer.
r - Exclude ranges - these ranges cannot be within the match part of the
    file. You can once again specify numerous ranges, all will be compared
    against the first 2048 bytes. For example, to figure out an plain text
    file you might try r001F7F9F, so any file with chars within 00-1F and
    7F-9F is passed through to the next preference entry.
    You need to specify always 2 bytes (4 digits).
g - Range which may exists at the current position - means same as
    v but gives a valid range instead. For example, g3039 will match
    any numeral digit at the current position.
    You need to specify always 2 bytes (4 digits).
s - Describes a string which has to be within the match part of the file,
    but at any given position therein. For example, you could detect post-
    script files using s25215053 ("%!PS").

examples:
v464F524D   describes an IFF file - matching FORM at the start
v464F524Dm8v494C424D  describes an IFF ILBM file - matching FORM at the
      start and ILBM at position 8
r00405BFF   matches any file having only UPPER characters A-Z
g415A617A   describes a file having an UPPER character at
      first and a lower character at second position
v000003F3l6401    Executable file with size up to 25KB
v000003F3h6400l19001  Executable file with size between 25KB and 100KB
v000003F3h19000   Executable file with size greater than 100KB
v50503230|v50583230 PowerPacker files: Header is PP20 or PX20

Example File Types:

Files marked with NoPack are already crunched.

Type        NoPack  File Pattern

DMS Archive    (DMS)    * v444D5321
Executable     (EXE)      v000003F3
GIF Picture    (GIF)    * v474946
HotHelpHeader  (HHH)      m6v486F7448656C70486561646572
HotHelpText    (HHT)    * m6v486F7448656C7054657874
```

| | | | |
|---|---|---|---|
| Icon File | (INFO) | | vE3100001 |
| ILBM Picture | (IFF ILBM) | | v464F524Dm8v494C424D |
| Intellifont | (IOF) | | v00440001m10v0014FFFF |
| JPEG Picture | (JFIF) | * | vFFD8 |
| MED Module | (MMD) | | v4D4D44g3039 |
| LhA Archive | (-lh?-) | * | m2v2D6C68m6v2D |
| LZX Archive | (LZX) | * | v4C5A58 |
| PNG Picture | (PNG) | * | v89504E470D0A1A0A |
| PT 32 Module | (PT MOD 32) | | m438v4D2E4B2E |
| PT Packed Song | (PT SONG) | * | v5041434B |
| Sound File | (IFF 8SVX) | | v464F524Dm8v38535658 |
| Startrek. Mod. | (ST MOD) | | m438v464C54g3039 |
| WordWorth Text | (IFF WOWO) | | v464F524Dm8v574F574F |
| XPK file | (XPKF) | * | v58504B46 |
| Zip Archive | (PK) | * | v504B0304 |
| Other IFF | (IFF) | | v464F524D |

## 1.8 XPK - A STANDARD FOR DATA COMPRESSION

MOTIVATION
----------

* Many programs that should offer data compression (e.g. HD backup utilities) do not.
* Many programs that offer data compression use old, slow, inefficient or inappropriate algorithm.
* All programs that offer data compression offer just one algorithm, you are stuck with that one.
* Many good packers are not used by any application program and have no good user interface.
* The installation of most packers requires AmigaShell knowledge (putting LhA in the path so that Directory Opus can find it)
* The decompression of all files packed with existing packers requires knowledge about the packer used for compression.
* Many compression programs can not deal with files that are larger than available memory.
* The existing compression programs are either slow or have a low compression factor.
* There is no way to support upcoming hardware compression cards in already existing applications yet.
* For none of the current compression programs exists a real decompressing file handler that uses no dirty tricks to decompress files on the fly.

The solution to all these problems is XPK.

OVERVIEW
--------

The xpk standard is to data compression what xpr is to file transmission.
It consists of three layers:
Level 2: The application/xpk interface for archives
Level 1: The application/xpk interface for files
Level 0: The xpk/packer interface
In addition, there is an optional standard xpk file format.

All parts of the xpk standard are implemented in shared libraries. There is one master library for archive level access, one master library for file level access, and one library for each compression algorithm.

```
level 2       xpkarchive.library
         |      |
         V      |
level 1       xpkmaster.library   |
        |    |     |    V
level 0 type 3      |    |     |  xarZOO
        |    |     V
  type 2      |    |  xexPowerPacker
        V    V
  type 1   xpkNUKE  xpkENCO
```

LEVEL 0 LIBRARIES
-----------------
All level 0 libraries offer the same functions. They are very small. Typical calls are: "Tell me what you can", "Compress this chunk of memory to another chunk of memory", and "Decompress this chunk of memory to another chunk of memory". These libs are very limited, their functionality is expanded by xpkmaster.library. No one would want to talk to a sub library directly.

THE LEVEL 1 LIBRARY
-------------------
Offers functions like "Compress this file to that chunk of memory using that algorithm". All combinations permitted: Mem to mem, file to file, mem to file, decompression and compression. Asynchronous packing possible. Very convenient tag based caller interface. Determines automatically out which sub library to use for decompression. Returns detailed error messages.

THE LEVEL 2 LIBRARY
-------------------
Offers archiving functions like "add this file to that archive" or "show me the contents of that archive".

OVERRIDING
----------
It is planned, that libraries of a lower level can offer higher level functions. They should be able to override the automatic functionality expansion by the higher level library. xpkmaster.library, for example, enforces the use of the xpk standard file format. It should be possible to override this by a sub library. Therefore an new library interface will be created, the xex libraries.

THE XPK FILE FORMAT
-------------------
Offers checksums, chunks (important when Seek()s on a compressed file become necessary) and automatic handling by the xpkmaster.library. This means that any new packer that can only pack mem to mem has its own file format immediately. And most important: The name of the packer library is contained in the file. Therefore, copying a new sub library to LIBS: is all you have to do to install a new packer (easily done in installation scripts); xpkmaster.library recognizes the new file type immediately. No changes to xpkmaster.library or the application programs necessary. In case the xpk file format is not used, the introduction of a new packer requires a change the xpkmaster.library.

```
TYPICAL APPLICATIONS
--------------------
A few examples for applications that could use xpk:
* A GadTools based archiver interface that can deal with all archivers
* A CLI based file compressor/decompressor [xPack, xPK, xUp]
* A hard disk backup utility that stores compressed data [Diavolo and
  others]
* A tool to write compressed images of devices to files [PackDev]
* A 'more' program with automatic decompression [Most, MuchMore]
* A DTP program that stores its fonts in compressed format
* A network protocol with built in data compression for slow connections
* A hypertext utility that allows all data to be compressed
* A file handler that overlays an existing filesystem and uncompresses any
  file while loading [XFH, PackDisk, Diskexpander (EPU)]
...plus many more we do not even need to think about yet.

CONCLUSION
----------
XPK would increase the usefulness and flexibility of both application and
compression programs while improving their user friendliness at the same
time. The best way to establish this standard would have been to distribute
it on the workbench disk that came with every Amiga.
```

## 1.9 gnu-license

## 1.10 Documentation of the included sub libraries

```
CBR0
  Yet another CmpByteRun0 algorithm compressor

DLTA
  A trivial delta preprocessor

DUKE
  A
NUKE
 variant tuned for sampled sound

FAST
  A fast LZRW based compression algorithm

FEAL
  A Fast Encryprion ALgorithm

HFMN
  A fast packing & unpacking dynamic huffman

HUFF
  A dynamic huffman cruncher/decruncher

IDEA
  ABPs IDEA implementation for XPK
```

```
                IMPL
                  A LZ77 variant supporting various compression modes

                MASH
                  Another LZRW based compression algorithm

                NONE
                  A dummy packer doing no compression

                NUKE
                  A LZ77 variant with fast decompression

                RAKE
                  A cruncher of the LZ77 family

                SHRI
                  LZARI variant

                SMPL
                  A dynamic huffman with delta precoding

                SQSH
                  A LZ based cruncher with special algorithms for 8 bit sample ←
                      data

Use
                xBench
                 for comparission between packers.

Legal issues:
These libraries may be freely distributed, as long as they are kept in
their original, complete, and unmodified form. It may not be distributed
by itself or in a commercial package of any kind without a written
permission.

These libraries are distributed in the hope that they will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE.

Most of them you can redistribute and/or modify under the terms of the

                GNU General Public License
                .
```

## 1.11 cbr0

```
                    Copyright 1992 Bilbo the first of Hypenosis

xpkCBR0.library is a standard XPK sublibrary implementing the very simple
cmp byte run 0 compression algorithm. The same algorithm is used on
compressed IFF-ILBM files. It is well known that this algorithm is only
efficient on data containing repeating equal bytes. This means that ASCII
files or (not compressed) picture files will be compressed well, but
executable files, sound data files, encrypted files (or other white noise
data) will be compressed only approx. 3%.
```

```
xpkCBR0.library is of course:

 · written 100% in 68000 assembler using DevPac V3.02,
 · reentrant,
 · pc-relative (except for resident structure used by system for injection),
 · some bytes shorter than
                U.D.Müller’s
                 RLEN,
 · 2.9 times faster on compression, 3.6 times faster on decompression
   compared to RLEN both used on file AmigaVision
 · written by Bilbo the first of Hypenosis (this fact should convince you)

Version History
1.0 First public release.
  No known bugs.
```

## 1.12 dlta

```
                FEATURES
-good when crunching modules/sounds in combination with a cruncher
-written in fast optimized assembly (joh mei)

DOCUMENTATION
The DELTA enciphering routines of xpkDLTA were developed to help
xpk-crunchers crunching SOUNDS and MODULES.

In this version xpkDLTA supports BYTE-Delta encoding. This is the most
efficient encoding algorithm in combination with samples. WORD-Delta and
LONG-Delta may follow if the first 16-BIT (32-BIT!?) samples pass my way
or if you ask me kindly.

Example:
8SVX-Sample, Music mixed with talking

Uncrunched:   1016484 byte 8SVX-Sample

Imploded:   789824 byte (77.7% left) IMPL.100ed 8SVX

Deltaed+Imploded: 628076 byte (61.7% left) DELTA+IMPL.100ed 8SVX

HOW IT WORKS
------------
xpkDLTA takes a byte and looks what the difference is between this
byte and its successor. It stores these differences. That is all!

Example:

|DATA |DELTA
|-------+-----
|6  |+6 (6-0) ;Start-Value => precedessor=NULL
|7  |+1 (7-6)
|3  |-4 (3-7)
|4  |+1 (4-3)
|10 |+6 (10-4)
```

FUNCTIONS: xpkDLTA supports all standard xpk sublibrary functions.

CONTACT
If you want an update, enclose enough DM (Deutsche Mark) for disk, stamps, envelope etc.

Never forget to mention
-what of my programs you are using
-which version
-where you got it from

Fanpost, donations, suggestions, ideas, flames & comments are welcome.

Get the authors address (Stephan Fuhrmann).

## 1.13 duke

DUKE is a hacked version of
NUKE
combining the effects of
DLTA
and
NUKE
.
Its compression performance and ratio probably is not good enough, we still need a good lossless sound and/or module packer.

## 1.14 fast

xpkFAST is an XPK compression sublibrary whose main purpose is to ↩ be
fast. The most interesting part of FAST is its speedy compressor, which makes it predestined for applications which compress about as often as they decompress. Good examples are: backup systems which make use of XPK to support compressed backups or compressing filesystems. An introductory text to the concept of the XPK compression system can be found in the OVERVIEW document supplied by the standard XPK distribution.

FAST consists of three parts, two compressors and a common decompressor. You can choose between the two compressors by using FAST.0 up to FAST.79 for the ``speedy'' compressor and FAST.80 up to FAST.100 for the ``crawling'' compressor, which is still faster than
NUKE
. The default mode
is FAST.50 which selects the ``speedy'' compressor.

```
Algorithm
---------
```

FAST is a member of the LZ77 family of datacompressors. Other popular

```
members of the LZ77 family are: xpkNUKE, PowerPacker, Imploder (xpkIMPL)
and some parts of lha, gzip, zip, zoo, freeze, arj, uc2, ha, ain, ...

The common thing about all LZ77 compressors is that they store the data
as sequences of <copy>- and <quote>-items. FAST uses one `control-bit' to
distinguish between a <copy>- and a <quote>-item. A <quote>-item simply
consists of one byte which has to be placed into the outputstream
uninterpreted. Each <copy>-item consists of 12 bit <distance>- and 4 bit
<length>-information. <distance> encodes where to copy _from_. The 4095
useful possibilities are 1..4095(*) bytes back in the outputstream.
<length> encodes _how_many_ bytes to copy. Possible <length>s range from 3
to 18, which are encoded as 18-<length>.

The input: aaaaadadada compresses to: Q(a) C(1,4) Q(d) C(2,5). Where
Q(char) is a <quote>-item and means write a single character 'char' to the
output and the <copy>-item C(dist,len) means copy 'len' bytes, which can be
found 'dist' bytes back in the output, to the output.

FAST uses two datastreams. That is, the compressed data consists of two
parts, the wordstream and the bytestream. The first compressor which used
this technique was xpkNUKE. The bytestream starts at the beginning of the
compressed data and the wordstream is stored in reverse order beginning at
the end of the compressed data. Thus the compressed data does look like
this: literalsSSDDRROOWW where small characters denote literal bytes and
two capital characters are a word from the wordstream.

If you want to discover more of the internal workings of xpkFAST just:
``Use the force! Read the source!'' The best place to start your tour
through the source is the decompressor in decompress.s since the
decompressor is much simpler than the compressor.

(*) I could have been using distances of 1..4096, but doing so would
  have added one instruction to the short and thus fast decompressor.

History:

In April 1991, Ross Williams published his LZRW1 algorithm by presenting it
at the data compression conference.

The LZRW1-A algorithm is a direct descendant of the LZRW1 algorithm,
improving it a little in both speed and compression.

FAST started as a ``port'' of Ross Williams' LZRW1-A C-Implementation
and his 68000-version of the decompressor to the Amiga as xpksub-library.
While porting I made some small changes improving the decompression speed.
I removed the feature of handling the case of noncompressable input,
because the xpkmaster.library takes care of that. After that, I found some
cute changes which dramatically improved the speed of the decompressor.
These were in detail:

 *  split the compressed data into a word- and a bytestream, removing many
    double byte accesses with a shift in between.
 *  changed the copy loop from a move-dbra loop to 18 moves in a row.
 *  changed the used range from 1..4096 to 0..4095 eliminating one
    instruction in the decompression loop.
 *  removed all bra.s from the inner decompression loop.
 *  totally rewrote the compressor in 68000 assembler.
```

```
    + changed the hashfunction to NOT use mul or div.
    + produces the ``new'' format needed by the new decompressor.
    + removed nearly all of the loop control tests by having
      a fast and a safe loop.
    + small code fits into the instructioncache of a 68030.


                Urban Dominik Müller
                 helped me to improve the speed of the compressor
even further, contributing several ideas and some code. For details refer
to the source.

V1.00:  release date: 29-Aug-1993

V1.01:  unreleased.  [testversion with four different compressors.]

V1.02:  release date: 12-Sep-1993
  * quadrupled the HASHSIZE for FAST.80 .. FAST.100 which allowed the
  removal of 2 now unneeded COMPARE_BYTEs to speed up compress_slow.

V1.03:  release date: 17-Oct-1993
  * major code juggling in compress2.s to squeeze some cycles.
  * removed the need for a ctrlCtr in compress2.s in favour of doing
  addx.w ctrl,ctrl  bcs.s ctrlFull and ctrl initialized to #1
  instead of rol.w #1,ctrl  dbra ctrlCnt,notFull and ctrl initialized
  to #$0000FFFF

V1.04:  release date: 06-Feb-1994
  * fixed a buglette reported by Detlef Riekenberg <eule@netgate.fido.de>

V1.05:  release date: 01-May-1994
  * removed MEMF_CLEAR from call to AllocMem() of the hashtable which
  is initialised anyway. reported by Simone Avogadro
  * cosmetic changes to compress2.s

V1.06:  release date: 28-Jul-1994
  * tuned the copying of the wordstream in compress.s and compress2.s
  * rewrote bitreading in the decompressor

"Thank you"s must go to:

    Jörg Bublath    <bublath@forwiss.uni-passau.de>
  for never getting tired of assembling and testing new versions.


                Urban Dominik Müller
                  for providing ideas and code to improve FAST, XPK itself
  and doing various xBenchmarks on his A4000 and A3000.

    Ralph Schmidt   <laire@uni-paderborn.de>
  for providing BAsm and BDebug  [In my opinion the best
  development environment for assembler programs on the Amiga.]
  and doing some batch-xBenchmarks on his A4000.

    Michael van Elst    <mlelstv@specklec.mpifr-bonn.mpg.de>
  for being so couraged to run one of the first alpha versions
  of the crawling mode on his A3000 during a large filetransfer
  --- and crash.

    Markus Illenseer    <markus@TechFak.Uni-Bielefeld.DE>
  for enabling me the remote-use [and once -guru] of his A2000+68030
  and temporarily ripping all the 16Bit FAST RAM out for the sake
  of acurate xBenchmarks.

    Tobias Walter   <walter@jazz.hall.sub.org>
  for letting me use his A1000 to test 11 totaly different and
  incompatible versions of FAST in one evening.


                Matthias Meixner
                  for doing some xBenchmarks when Jörg was ‘unavailable’.

    Markus Armbruster   <armbru@pond.sub.org>
  for assisting me in the two weeks search for the
  _nonexistent_ timing-indeterministency-bug.

Contact Addresses:

    Ross Williams
    ross@spam.ua.oz.au


                Christian von Roques

                Urban Dominik Müller
```

## 1.15 feal

```
                    xpkFEAL is an XPK encryption sublibrary which implements the  ↩
                       FEAL-N data
encryption algorithm in CBC1 mode.  FEAL-N has been developed at the NTT
Communications and Informnation Processing Labs. in 1988.

    FEAL-N is a blockchifre, which encryptes a datablock of 64Bit to a 64Bit
codeblock using a 64Bit external key.  FEAL mainly consists of a loop which
is taken N times.  The loopbody encodes half of the data using a 16Bit
internal key and swaps the encoded half with the other one.  The 64Bit
external key is expanded to N * 16Bit internal keys.

    FEAL was designed to be a replacement of DES.  DES can be easy made fast
using special purpose hardware, but is a pain to be implemented in software
using conventional hardware.  Since FEAL only uses 8Bit add, rol and eor
operations, it is designed to be implemented in software.

    (Btw.: FEAL is one of the few algorithms which is easier to implement
using the 80x86 processorarchitecture than the 680x0 because of the 80x86s
splitable registers.)

           Safety
           ------

    Rounds  Safety (? ;-)
```

```
   ------  -------------
    4  unsafe, broken (Murphy 1990)

    8  unbreakable for ``normal'' people

   16  good Cryptoanalysists can decypher this with less
      (default)  then testing all possible keys.  But it can be valued
     as ``safe'' anyway.

   32  There is no known better method of breaking this
     than testing all 2^64 possible keys.

   64  Only paranoids will use this.
     ( But real paranoiac do not use FEAL )

           History of FEAL
           ---------------

   1985  first proposal to ISO ( FEAL-1, FEAL-1', FEAL-2 )
   1987  FEAL-4 presented on Eurocrypt.
   1987  attack on FEAL-4 by B. den Boer. ( Crypto 1987 )
   => doubled the number of rounds: FEAL-8
   1988  FEAL-N proposed (N even >=4)
   1988  FEAL-NX proposed (N even >=4)
   different method to calculate partial keys
   => 128Bit key instead of 64Bit


          published attacks
          -----------------

  o B. den Boer (1987: FEAL-4; 100-10000 choosen plaintexts)
  o Murphy (1990: FEAL-4; 4 choosen plaintext)
  o Gilbert Chasse (1990: FEAL-8; statistically)
  o Bilham, Shamir (1990: FEAL-4. FEAL-8, FEAL-N, FEAL-NX)
    differencial Cryptoanalysis:
    => for up to 31 rounds better than testing all keys.
  o Gilbert (1991: FEAL-4, FEAL-6; 20000 knowm plaintexts)

      published versions of FEAL
      --------------------------

   name     rounds key   internal key
   ----     ------ ---   ------------
   FEAL-1  4  64 4*16+2*32

   FEAL-2  6 128 6*16+2*32

   FEAL-1'
   FEAL-1.00 4  64 4*16+2*64
   FEAL-4

   FEAL-2.00
   FEAL-8  8  64 8*16+2*64

   FEAL-N  N  64 N*16+2*64
   FEAL-NX N 128 N*16+2*64


        Version History of xpkFEAL
        --------------------------

   1.0 First public release.

   1.02  Fixed a stupid typo, which did not prevent the user from
     encrypting with an uneven number of rounds.

   1.03  Previous versions filled the last block with junk, now
     the last encrypted byte is length&7.
     Minor speedups in the assembler part.


         Future Plans
         ------------

     Support the other 3 standard modes. (ECB, CFB and OFB)
     Improve the speed.


             Contact Address
             ---------------


                 Christian von Roques
                   +-----------------------------------------------------------+
   |  Questions regarding FEAL-N can be referred to:    |
   |    Mr. Shoji  Miyaguchi          |
   |    Communications and Information Processing Labs., NTT  |
   |    1-2356, Take, Yokosuka-shi, 238-03, JAPAN       |
   +-----------------------------------------------------------+
```

## 1.16 hfmn

```
               This XPK sub library basically uses the same algorithm (dynamic ↩
                  huffman or
classic huffman) as found in the xpkHUFF.library. For more detailed
information about the huffman algorithm, take a look into HUFF.doc from
M.Zimmermann -- and skip the part that huffman compression & decompression
is pretty slow! In difference to HUFF, HFMN is FAST on compression and
decompression and produces a slightly better output. Although the basic
algorithm is the same, it is entirely different implemented, therefore
HFMN will not depack HUFF and HUFF not HFMN!

     HFMN needs for private buffers (no xpkmaster.library buffers)
```

- 7.5 Kbyte packing memory
- 5   KByte unpacking memory

How does it work?

- First, I use heapsort to create the huffman tree, which is most responsible for packing speed. (heapsort is the second-best sort algorithm and is based upon binary trees)

- Second, (for decompression) I generate an (almost) optimal unpack code from the huffman tree.

- Third, I save the huffman tree recursivly. That is why I need max. 320 byte to save a complete huffman tree.

```
      020+ Version
      ------------
```

I have experimented with 020+ code and rewrote the most used routines. My huffman-code-translation-routine :) is reduced from 50 to 16 instructions, and achieves a noticable speedup. (hmm, I like bitfield instructions.:-)

```
.next move.b  (a0)+,d2    ;incoming characters ( $00-$ff )
  move.l  (a2,d2.w*4),d3    ;huffman code
  move.b  3(a3,d2.w*4),d4   ;huffman codesize
  bfins d3,(a1){d5:d4}    ;store huffman code
  add.l d4,d5     ;bitoffset + last codesize
  dbra  d7,.next
```

For decompression, the 020+ code produces no improvements. But there were still some small optimizations possible, so decompression speed is improved too.

```
        Version History
        ---------------

    V 1.16 - first public version.
    V 1.18 - 2 ways of decompression.
    V 1.19 - bugfix: uncompressable data returns now XPKERR_EXPANSION
       instead of XPKERR_SMALLOUTBUF.
 V 1.20 - V 1.27 - some experimental versions with 020+ code.
    V 1.28 - extra library with 020+ code for compression.
     - improved 000+ decompression code.
 V 1.29 - V 1.33 - some experimental version for the 1.34 bugfix.

    V 1.34 - fixed a bug that i had added somewhere before 1.16.
       it should have caused problems only under bad circumstances,
       when the byte statistic was fibonacci like.
       (in fact, the decompression routine could not handle
        huffman codes longer than 16 bits, ups...)
       Thanks to Nicolas Pomarede for his superdetailed bugreport.
       (He analysed the code and told me exactly when and where it
        goes wrong :-) )

    V 1.35 - fixed a bug in the 020+ compression routine.
       (16 Bit overflow for number of bytes written to xsp_OutBuf
        was not handled correctly)
       Thanks to David Balazic for reporting this one.
    V 1.36 - 1.35 bugfix was not 100% ok.


        Contact Address
        ---------------
```

Martin Hauner

## 1.17 huff

The idea of a huffman crunch is as follows: often used bytes (ie 8 bit
codes) get a shorter code than 8 bits, fi 5 bits. So everytime one of these
bytes occurs in the source file I save (in this example) 3 bits in the dest
file. To find out the optimum codes for each byte there is a simple method:
A tree is to be constructed so that the path from the root to each leaf
reflects the optimum (ie huffman) code. Unfortunately most computers (the
Amiga, too) are byte-oriented, which means a rather complex handling of
codes that are not a multiple of 8 bits. This results in pretty slow
compression & decompression. So this means that the xpkHUFF.library
probably will not be used for normal circumstances, but, as Dominik stated,
it may serve well as an example library.

There are three different huffman crunch algorithms known:

- static compression/decompression
- dynamic compression/decompression
- adaptive compression/decompression

What are the differences?

The static huffman uses a fix table to represent each byte of the source.
This, of course, makes sense only, if the structure of the data to be
crunched is known. In this case (for instance crunching an english text) a
fix table of codes is embedded in the code. Crunching other data than what
the table was generated for will probably result in worse compression or
even expansion.

This is what a dynamic huffman is avoiding: it first creates a statistics
table reflecting the frequency every byte occurs with and generates a
special tree/table for this case, so the dynamic huffman does a good
compression for this special data.

But there is something that can be improved, anyway: imagine, there is a
data block which contains many 'a's in its first part and many 'z's in the
last part.... The dynamic huffman would represent the 'a's and 'z's with
short codes, of course. But it probably would be even better if the
crunch/decrunch tree would reflect the *current* data beeing processed
instead of the whole block, thus in resulting shorter codes for 'a' and 'z',
depending of the position in the data block. This is what an adaptive
huffman deals with: it creates the tree while crunching, without doing any
statistics or tree creation before crunching. It permanently updates its
internal huffman tree. Therefore it does not have to include the information
about the decrunch tree in the crunched data.

Final words about huffmans:
A stand-alone huffman will never achieve crunch results as fine as those
reached with most other crunchers, for these will not only regard the number
of occurances for each byte (as huffman does), but sequels of byte, too.
This means: If you create all permutations of a datablock, the huffman

crunched will always have the same length. Others will not, as they are depending on the order of the crunched data, too.

## Description

The library 'xpkHUFF.library' implements a dynamic huffman crunch algorithm, even though the adaptive might result in slightly better crunch results. However, this is more complex to implement and I am using a maximum buffer size of 64K, so this is a little bit like an adaptive huffman for large files.

If I should have lots of spare time I will probably implement an adaptive huffman crunch algorithm. This new library will be called xpkHUFF, too, and new xpkHUFF.libraries will always handle output generated by earlier versions.

The xpkHUFF.library supports a totally unsafe (but a little bit better than simple eor :-) encryption. Please note that crunch/decrunch speeds decrease when encryption is used.

## Implementation

If you should see an errormessage saying output buffer too small while crunching *and* encrypting, this means you tried to crunch and encrypt a file that would crunched and encrypted be larger than the original file. This should occur only with very small files (for I have a minimum file size due to tables) or with files that have been crunched already and therefore would expand during crunch.

A technical note: this could also happen, if the last chunk of a file to be crunched/encrypted would be dimensioned too small by xpkmaster.library.

However, in this case you cannot encrypt the file. I know this could be annoying and will think about a solution for this problem, but remember: this encryption would not be safe, better if you used FEAL or IDEA for secure encryption.

## Last words ...

I tried hard to debug this library with range checking while writing bytes on crunching, and so on, but as in every code larger than, say 10 lines :-), there will be bugs. I do not know any bugs in this version, but if you should meet one, please let me know via email. As usually, reproducable bugs are preferred. Please add your configuration, programs running (best if you try without startup-sequence!), and, most important of all, add the file you tried to crunch! Thank you.

## Version History

```
; V 0.1  - 12-Jul-1992 :  first version
; V x.yy - 18-Jul-1992 :  first OK version
; V x.yy - 19-Jul-1992 :  sped up decrunching
; V x.yy - 21-Jul-1992 :  bug fixed in word/long decrunching: min pack
;       chunk size now 3/5
; V x.yy - 21-Jul-1992 :  replaced many subq/bxx with dbf (ie sped up
;       crunching a little bit), bug fixed: there was
;       a dbf counter wrong (one of my favorite 0/1
;       problem bugs)
; V 0.50 - 29-Jul-1992 :  added 68030+ cache optimized decrunch code
; V 0.51 - 01-Aug-1992 :  byte decrunch improved, first code added,
;       indicator byte for crunchmethod used added,
;       68030+ chache optimized code does not make
;       sense any more, since byte decrunch fits to
;       cache completely, now
; V 0.52 - 01-Aug-1992 :  unsafe encryption supported
; V 0.53 - 03-Aug-1992 :  slight improvements made to crunch code
;       (+ 6K/s)
; V 0.54 - 03-Aug-1992 :  inconsistence in expansion handling fixed
; V 0.55 - 03-Aug-1992 :  bug fixed: expansion handling now considers
;       table creation, too
; V 0.56 - 03-Aug-1992 :  bug fixed: HUFF now can crunch files
;       consisting of always the same byte (shame
;       on me, termination criterium was wrong)
; V 0.57 - 03-Aug-1992 :  Tree creation code partially rewritten
; V 0.58 - 05-Aug-1992 :  bug fixed: wrong termination criterium for
;       expansion check (my favorite 0/1 problem)
; V 0.59 - 06-Aug-1992 :  now decrypting in a special buffer, not using
;       InBuf (this is read only, I was told) any more
; V 0.60 - 07-Aug-1992 :  added extra memory required during
;       packing/unpacking
; V 0.61 - 08-Aug-1992 :  expansion check changed, renamed from
;       xpkDHUF.library to xpkHUFF.library thus
;       corresponding to the possibility of handling
;       adaptive huffman codes later without having
;       an inconsistence in the name
; V 0.62 - 10-Aug-1992 :  Flag XPKIF_MODES removed (I do not have modes
;       yet (but I have a mapping code :-=))

Contact Address:
                Marc Zimmermann
```

## 1.18 idea

```
          Patent
          ------
       IDEA is registered as the international patent  WO  91/18459
       "Device for Converting a Digital Block and the Use thereof".
        For commercial use of IDEA, one should contact

             ASCOM TECH AG
          Freiburgstrasse 370
       CH-3018 Bern, Switzerland

        Description
        -----------

IDEA is an XPK packer sublibrary which implements a highly optimized form
of the IDEA encryption algorithm.
```

IDEA (International Data Encryption Algorithm) is a block cipher developed by Xuejia Lai and Prof. Dr. J. L. Massey at the Swiss Federal Institute of Technology. See patent for information on any commercial use of this algorithm. Especially, this library is not only claimed by the copyright of the author and the copyright of the author of the used IDEA kernal routine, but by the copyright of the IDEA originators and their patent, too.

This implementation of the algorithm was done by André Beck, Dept. of Computer Science, Technical University of Dresden, Germany.

xpkIDEA.library gives a chunk based access to the most common encryption methods, using the IDEA cipher as the encryption function. The IDEA cipher is known to be somewhat slow. It performs 34 multiplications modulo 2^16+1 for every 64 bit data packet, so it must have limited performance on a plain 68000 processor. This library uses the heavily hand optimized, permuted, macrotized and partially unrolled 68000 assembler implementation of IDEA by Colin Plumb. Therefore, the kernal IDEA routine and it is macros are copyright by Colin Plumb.

In difference to the most wide spread compressors distributed with XPK, one should know something about IDEA before using it. First, IDEA is completely no compressor, it only encrypts or decrypts data. A password must be specified with first calls to "pack" or "unpack" a chunk. Furthermore, the password given on encryption MUST restore the original chunk contents, otherwise the password will be treated as incorrect. This is a tribute to the XPK architecture and its safety, but has the disadvantage of preventing you from doing things like a triple crypt, what means to first encrypt a chunk with password 1, then decrypting it with password 2 and last encrypting it with password 3, all three passwords different.

## Encryption Methods

IDEA is a cipher used for encryption in this library, what means it is a function taking one data block and an encryption key as input and producing one data block as output. The purpose of this function is to generate a very random result from normaly highly redundant input, in other words to make White Noise of bits from a regular, low entropic bit stream. The IDEA data blocks are sized 64 bits, where the key has 128 bits in its unexpanded form (DES has a key of 56 bits). One now may use this function in different ways. The simplest encryption is to take the input chunk block by block, driving it through the IDEA function, and building the output chunk from the result. This mode is called Electronic Code Book (ECB). But an Code Book based encryption is not the state of the art, because it is somewhat easy to crack (even ECB using IDEA is not easy to crack, only a bit easier than the following modes). One can imagine, that including the chunks contents (which is to be crypted) itself into the encryption will be much safer. Consider a simple ECB to encrypt text, generated by the function

```
 out_character = (in_character + 1) MODULO num_of_characters.
```

This is nothing other than incrementing every character, f.i. making A to B, F to G and Z back to A. So the word FOOBAR will be crypted to GPPCBS, and nobody will see what it is on the first visit. But there are also people called Cryptologists, and cracking such codes is their job. Simple methods of cracking are especially based on the probability of characters in different languages. They know e is a very often found letter in indo european languages, and if they find one character very often in the crypted text, this one may be an e. If it is sure that it is an e, one can insert it in the complete crypted text where the cracked character was.

The method to prevent such simple cracks is based on chaining the produced output back into the crypt function with some delay.
Consider

```
 out_character = ((in_character + last_out_char)+1) MODULO num_characters
```

with an initial last out character of 'C'.

FOOBAR gets JAQTVL using this code and nobody can see that an double O was in the input. So it is more complicated to crack messages crypted with this code, because one MUST start at the beginning of the text. It is also possible to increase the number of ,,states of remember'' we are using, for instance by not using the last_out_char but the seventh_last_out_char and using 7 different initial values for them.

The method used above is very similar to a common encryption method called Cipher Block Chaining (CBC) with one state of remember (CBC 1).

The difference to ECB in schematic view:

```
ECB electronic code book mode
    y[i] = IDEA(z, x[i])
    x[i] = IIDEA(z, y[i])

CBC cipher block chaining mode
    y[i] = IDEA(z, x[i] ^ y[i-N])
    x[i] = IIDEA(z, y[i]) ^ y[i-N]
```

with

```
  x[i] is the input block number i
  y[i] is the output block number i
  z    is the encryption key
  N    is the number of states of remember (at least one)
  IDEA is the encryption function using the IDEA cipher
 IIDEA is the corresponding decryption function
  ^    means the XOR of the operands (Bitwise Exclusive Or)
```

There are two additional modes often used with encryption. See the following schematics:

```
CFB ciphertext feedback mode
    y[i] = x[i] ^ IDEA(z, y[i-N])
    x[i] = y[i] ^ IDEA(z, y[i-N])

OFB output feedback mode
    h[i] = IDEA(z, h[i-N])
    y[i] = x[i] ^ h[i]
    x[i] = y[i] ^ h[i]
```

As you see, all the chaining modes have additional parameters determining the

```
result of the crypt. Not only the key determines the resulting chunk for a
special input chunk, but also the number of states of remember used by the
mode and the values used to initialize the states (in CBC 1 coding block
#0, you need block #[i-1], but you have no block #-1, so you have to give
some initial value for it). For CBC 8 you have to give 8 initailizers,
and so on.

The xpkIDEA implementation uses the following XPK modes for different
encryption methods:

 0.. 25 ECB
26.. 50 CFB
51.. 75 OFB
76..100 CBC

As you see, the modes were ordered to somehow match the scheme given by the
most XPK packers, with 0..100 mapped to increasing efficiency and decreasing
speed. There are neither big differences in speed nor in efficiency of the
used modes, and the mapping used is easy to remember. Especially one gets
very simple from the mode used to the encr. method and state number by
subsequent subtractions of 25 from the mode:

  IDEA.79 -> 79 - (3*25) = 4 , so mode 3 (CBC) with 4 states applies.

The default method used when no mode is explicitely given is CBC1,
i.e. the mode IDEA.76

The presented speed (in KByte/second) is not very exact. This is mainly
caused by the varying cycle count of the 68000's mul instruction. The
encryption will be faster with the all-zero-key and slower with the
all-ones-key. Try around with key values
 #0
 #5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a5a
 #ffffffffffffffffffffffffffffffff
to see the differences.

Not only IDEA.100 is a very safe encryption, also IDEA.75 and IDEA.50 may
be good for safe results. They are modes with 25 states, so one may give
25 (!) different initializers to the password, which must all be known to
get this decrypted again. The code is developed in a way that no speed
loss will occure even using much states. At the other hand, a open connection
with this sublibrary for packing or unpacking forces the allocation of around
600 bytes of memory. If you are low on memory, the library may return a
matching error condition.

             The Password
             ------------

You may ask now, how to give different initializers to the encryption
modes which use them ? Therefore, the password parsing routine within this
library is more complicated than normal ones. An IDEA password consists
of the key value and a optional set of initializers, both specified either
as a plain ascii string to be hashed or as the explicite hexadecimal value.

The syntax is as follows:
 <password> ::= <keyspec>[<initializer>]*.
 <keyspec>  ::= <valuespec>.
 <initializer>  ::= ":"<valuespec>.
 <valuespec>  ::= [<charstring>|<hexstring>].
 <charstring> ::= ["!".."~"]*.
 <hexstring>  ::= "#"["0".."9"|"a".."f"|"A".."F"]*.

 so possible passwords are f.i.:
 password
 password:heut:ist:montag
 #738494ad53ae2c1b736218ac12abaacc:nix:hexa:oder:doch:#4455663311223311
 :      <-- this results in the all-zero-key.
 passwd::::ini4   <-- initializers 1..3 are zero

 Its useless to specify any initializers with ECB
 Its useless to specify more then N initializers for mode [CBC|CFB|OFB] N
 The maximum number of initializers is 25
 charstrings may have any number of characters
 hexvalues for keyspec have to fit in an OCTAWORD. (16 Byte)
 hexvalues for initializers must fit in a QUADWORD. (8 Byte)
 unspecified values (key/initializers) are zero.
```

If you do not initialize a value, it will be zero. Any syntactic or semantic error in the password specification will raise the error XPKERR_WRONGPW. The '#' character is used to introduce hex values because many shells would missinterpret $ even if it appears in doublequotes.
The hash routine currently used in this password parser is not very strong. String passwords should be at least 12 characters long to give a nice key.

## Technical Info

This lib is completely written in assembler using a68k and the 1.3 includes. It was developed within around 10 hours of work distributed over more than 14 days (better to say nights).
The author could only test it on an 1 Meg chip no fast 7.14MHz 68000 A500 under Kickstart 1.3. The source is now around 30000 bytes and may contain some bogus. If you find any bugs report them to me via the email address given below.
Make sure the output buffer is at least the size of the input buffer plus XPK_MARGIN, even if this is on decompression more then the original chunk size. This library relies on this behavior, which is correctly done by xpkmaster.

As already stated in the section Disclaimer, the author gives no warranties for the proper function of this software. Additional, he cannot give any guarantee that IDEA itself is a useful encryption standard. It SEEMS to be very strong, but it is still under analyzation by some organizations like the NSA and similars. If you are interested in the theoretics of this algorithm, ask me for some hints.

Contact Address:
See section Patent for information on how to reach the authors of the IDEA cipher.

If you want to get in contact with the author of the fast idea routine used

```
within this library, contact Mr. Colin Plumb at: colin@eecg.toronto.edu
```

## 1.19 impl

```
                This XPK sub-library uses basically the same algorithm as found in ↩
                    the
Imploder, but without the specifics needed for compressing self-contained
executables.

A quote from the Imploder 4.0 technical manual says it all :-)

IMPL does LZ77 like compression with a, per mode, static Huffman like
coding step on the various parts of the skip, offset and length tuples.
Due to the efficient encoding, a tuple can require less than 12 bits, and
thus strings of 2 bytes length and up are encodable with a decent gain
(given small Huffman patterns corresponding to likely circumstances).

The default compression mode is 100 which means that the actual compression
mode used depends on the chunksize. The default chunksize is 64K. In
general, this mode produces the best compression ratio, although the mode
range 76..98 (1.00*max) will sometimes produce better results.

The current version of xpkIMPL.library will, by default, react to a BREAK
signal (CTRL-C) while compressing.  Compressing a chunk (especially on
unaccelerated Amiga's) can take quite a bit time, so allowing the user to
break-off compression is useful.  For now, it is not possible to turn this
feature off!

Version History:

0.01 Well known Imploder compression algorithm now included in a xpk sublibrary.
0.19 Improved robustness. Released with xpk 2.4.
1.00 Much needed documentation added. ;-) Released with xpk 2.5.

Contact Address:
                Peter Struijk
```

## 1.20 mash

```
                xpkMASH is an XPK compression sublibrary whose main purpose is to
decrunch fast and have an excellent crunch factor.  The sublib is using
LZ77 compression and a special method to write matches...  MASH now
normally uses 256KB for its tables, but reduces the size of the
hashtable if memory is scarce.  (it could crunch even with 64KB+4KB)
Compressing with a small hashtable naturally is very very slow.
Default chunk size is 64KB.  The compressor uses lazy match evaluation
which slowed it down quite a bit.

This sublibrary has several modes:

  Mode        Strings to be searched
  ------        ------------------------
```

```
    0-09    1     ;high speed - but low CF
   10-19    2
   20-29    4
   30-39    8     ;good for most executables
   40-49        16
   50-59        32
   60-69        64
   70-79       128     ;this should be used for text files
   80-89       256
   90-99       512
  100      1024     ;the best, the slowest

The second column is showing, how many matches should be compared
- the more searched strings - the better results you will get.
formula is simple 2^(MODE/10).
MODE 70 is now runing as fast as NUKE on my A1200
and if you want to use some higher modes - you will get result better
for only a few bytes, but slowdown will be very noticeable.
(But for crunching I am always using the best mode anyway :-))

  !!! The source for this version is not released !!!
if you want to see it anyway, drop me an e-mail and I will send you it.

I still want to do some improvements. Probably even change format of stored
data to reach better decrunch speed and possibly use some more MC68020
instructions in this case. You do not have to worry, this library will also
decrunch old format. Send me an e-mail what you would like to see in newer
version of this library. But this newer format will always need
256KB of memory so it could be a problem for some people.
If you think this library is worth some money, you could send them
It will speed up development :-)

         "Thank you"s must go to:
         ------------------------

                Urban Dominik Müller
                   for XPK standart. (Try to respond to my e-mails sometimes :-))


                Christian von Roques
                   for correcting some parts of this document file,
   and also for releasing his source, so I could use some parts
   of it in my library (xpk interface).


                Karsten Dageförde
                   for making benchmarks and other cooperation

more people should be in this list - authors of Zip, Lha, Arj, ...
but I would have to make some deep research for them.

           History
0.5 Many errors, the biggest problem was bad writing of bits string.
0.7 Most errors have been debuged
0.8 Last byte has not been saved
0.9 On the first look, normaly working version of the sublibrary with
  fixed hash table - size 64KB
```

```
1.0 The big improvement in memory allocating;
  memory is allocated before each chunk compression and deallocated
  after this chunk is compressed (usefull if you have installed
  statram.device)
1.01  Hash size was increased from 64KB to 128KB (16 bits)
1.05  Hash is allocated dynamicaly - when is large memory free - large hash
  is used. Starting with 128KB, 64KB, 32KB, .... ,512 bytes
1.15  Seems to work perfectly for me

First public release:
1.16  I suppose last bug has been removed - value of register D4
  was not saved on return. Also most long word instruction have
  been rewritten to word oriented instructions (useful for MC68000)
1.26  Several speed up improvements - decompression goes about 50 kB faster

Unreleased
1.30  New crunch mode - uses 256KB of memory for its buffers
1.40  Removed checking of two bytes in match
  it is not needed when two-byte hash is used
1.53  Removed zero length write when chunk is uncrunchable
  Diavolo is a little bit odd and uses this value for DIVS
  even when its not valid -> caused GURU
1.61  Removed bsr call from scanning routine.

Released
1.77  Prepared for release - there are still many things to improve,
  but it already has a very good speed. So I am releasing this
  version.
1.98  Well many new checks have been added to prevent a to deep scan
  when better match cannot be found. Even a little bit better
  alghorithm was used for lazy_eval -> better CF.

Contact Address:
                Zdenek Kabelac
```

## 1.21 none

```
NONE is only a dummy packer, which was an programming example in the
first distribution. It only copies the data to the resulting file.
(With 52 or 53 bytes header)

It may be useful with xpkarchive.library, because it gives the option of
no crunching like in LhA.
```

## 1.22 nuke

```
                NUKE is an XPK packer sublibrary which implements a highly ←
                   optimized form
of the popular LZ77 compression algorithm.  This is essentially the same
algorithm used in PowerPacker, Imploder and (among other algorithms) in
the LZH/LHA packers.
```

Most applications of packers mean packing once and unpacking many times. One example is a PD program that gets distributed around the world, or a compressed program on the hard disk the needs to be decompressed when loaded. NUKE tries to~be fast at decompression, thus restricts itself from applying fancy algorithms (Huffman, Ari-coding). In order to achieve reasonable compression factors anyway, it scans a very long range (more than 24K) for identical byte sequences and if it finds any, it outputs offset and length instead of the bytes themselves.

Of course scanning such a long range for duplicates is quite a CPU intensive process. I have tried to make it as fast as possible, and with around 35K/sec (A3000) I would say I have come close to the best that can be done with this approach. There is a drawback, though. The compression must use large hashing tables to reach this speed. I have made sure that NUKE is still usable on a plain 512K Amiga, but you will not be able to run many things besides NUKE while you are packing. There is, by the way, no increase in mem needs with increasing file size.

```
Version History:
1.0 First public release.
1.2 Does compress slower, but a bit better.
  Decompression is faster than V1.0.
1.3 Fixed excessively long 2 byte matches [there were files, on which
  NUKE was not bijective!]
1.4     added new startup header
1.5     recompiled to fix error with DiskExpander
1.6 better library start header
1.7 fixed low-mem error
```

```
Contact Addresses:
                Urban Dominik Müller
                ,
                Christian von Roques
```

## 1.23 rake

```
                RAKE is an XPK packer sublibrary which implements a highly ↩
                   optimized form
of the popular LZ77 compression algorithm. It uses static huffman coding
for the 'len' and a three-step coding for the 'offset' information. The
major feature of this packer is the highly optimized algorithm for tracking
down redundant data.

RAKE supports four modes at compression (see below).
 - Scanner & Coder together fit in 68020 instruction cache
 - Hashbuffer-size will be reduced downto 0.5Kb, if memory is short

        History
        -------

Sep-9-94   V1.0 First public release (68000,68020)
     [...]
Jul-1-95   V1.6 - Scanning algorithm improved.
Sep-6-95   V1.7 - 68020 version now checks if there is an appropriate
      processor (68020 or better)
```

```
Contact Addresses:
                Karsten Dageförde
```

## 1.24 shri

```
                SHRI is an XPK packer sublibrary which implements a compressor,  ↩
                   highly
optimized for compression rate. It uses offset/len encoding with
adaptive arithmetic aftercoding for best compression results. Its
compression rate is better than that of most other packers, e.g. lha,
zoo or powerpacker.

    It supports 7 modes, which differ in the size of the dictionary:

   Modes   Dictionarysize
   ------  --------------
    0- 14     1k
   15- 28     2k
   29- 42     4k
   43- 56     8k
   57- 70    16k
   71- 84    32k
   85-100    64k

A larger dictionarysize gives a higher compression rate and faster
decompression, but slows down compression.

This library is not meant to be used for online-compression as it is used
in e.g. XFH. It is meant to be used for achieving highest compression-rates.
These can be useful, when transferring files via modem, for a backup to a
slow medium (floppy disks) when you have a fast computer or for creating
archives with the xpkarchive.library.

The decompressionspeed of this version is about 82% higher than that of
version 1.0. The compression is about 5% faster.

          History
          -------
V0.2  First public Release
V0.3  XpksPackChunk retured XPKERR_CORRUPT instead of XPKERR_EXPANSION
      - Bug fixed in V0.3
V0.4  SHRI could not decompress files, which where partly compressable
      - Bug fixed in V0.4
V1.0  New Version with improved compression- and decompressionspeed
V2.1  Improved decompressionspeed, all relevant parts of the decompressor
      are now written in assembler.
V2.2  Removed important bug in the decompression-part, that produced errors
      with chunksizes above 32767 bytes

Contact Address:
                Matthias Meixner
```

## 1.25 smpl

```
                SMPL is a XPK sublibrary implementing dynamic huffman coding over
variations of datastream. If that sound too complicated, I suggest you
read docs for
                DLTA
                 and
                HUFF
                , in that order. In fact, DLTA was made to be
used as preprocessor for other XPK packers.

Then why did I code SMPL? Think this: how many music programs you know that
support XPK? Yes, I know I can always use XFH so I can pack all my
data, but if I have first fed data thru DLTA and then another compressor,
then XFH only decompresses the latter. So I still need XPK supporting
program to pack my samples efficiently.

SMPL overcomes this by including
                DLTA
                 coding into same library. I chose
to use huffman coding for actual packing as it seemed to give best average
compression. I snatched the huffman code from
                xpkHUFF.library
                 and
tweaked it a bit for faster (de)compression.

Some samples were packed better with simple
                HUFF
                 without delta precoding.
If I find a way to determine output size from frequency table (ie. without
building huffman tree) I will add non-delta packing to SMPL.

I tested
                DLTA
                +SMPL mainly to see if there would be any use for recursive
delta, but less than 100K of all data packed marginally better when fed
thru double delta.

Three percent difference between SMPL and
                DLTA
                +
                HUFF
                 comes from two
things:
 1) xpkmaster.library adds some bytes to DLTA coded files
 2) I store huffman tree in more compact way

Version History:  1.00  First public release.

Contact Address:
                Jorma Oksanen
```

## 1.26 sqsh

SQSH is an XPK packer sublibrary which implements an optimized LZ ↩
based
algorithm combined with a 8 bit delta compression algorithm.

This packer was especially made for packing 8 bit Samples and ProTracker
style modules. It is NOT a lossy compression library, so NO quality-loss
will occur when packing Samples with this library.

SQSH is pretty fast at decompression (300K/s on A3000) so is very well
suited to compress Modules and Samples since these will typically be packed
once and unpacked many times. It is slow at compression (25K/s on A3000)
mainly because every part of the file has to be checked twice to see what
the better compression method would be.

In order to achieve reasonable compression of other types of files
(Executables, Textfiles) this packer will scan a long range (about 20K) for
identical byte sequences and if it finds any, it outputs offset and length
instead of the bytes themselves. Scanning such a long range for duplicates
is a CPU-intensive process. I have tried to make it as fast as possible
(about 25K/s on A3000) but
NUKE
proves it can be done faster :-)

In the archive also is included a 68000 version of this library. Sorry
all you 68000 users for the long delay, but I only received one message
asking me for a 68000 version. It was Edmund Vermeulen who eventually

Contact Address:
John Hendrikx

## 1.27 c-utils

xBench
benchmark tool for XPK sub libraries

xDir
directory lister

xLoadSeg
LoadSeg function patch

xPack
XPK packer and unpacker for OS2.0

xPK
XPK packer and unpacker for OS1.3

xQuery
XPK sub library information

xScan
scanner command for XFH

```
                xType
                    XPK type command

                xUp
                    XPK unpacker

  Program histories are in source code.
```

## 1.28 xbench

```
                SYNOPSIS
    xBench FILENAME/A,PASSWORD/K,METHOD/M,TEST/S,ALL/S

  FILENAME Is needed always. This is the name of the test file(s).
     You can specify names containing normal dos patterns.
  PASSWORD When given, xBench tries all crunching methods with and
     without password (when possible).
  METHOD   Here you may specify crunchers (multiple when needed). The
     format is <packername>[.<mode>] When no METHOD keyword is
     used, all packer libraries are scanned.
     Examples: NUKE, MASH.20, RAKE.100
  TEST   When you give this option, the decrunched buffer is
     checked against the source file, errors are reported.
  ALL  This option scans all modes, when no special one is passed.
     >>>> Use this with care, as there are 101 modes! <<<<
  SAVE   Here you may specify a directory, where all packed data is
     saved to. When nothing is specified, the data will not be
     saved.

  When no METHOD keyword is passed, all possible types are used.

  NOTE: When no password is given, all libraries needing passwords
  are skipped. When password is given all libraries are called
  with and without password. The real output depends on the fact, if
  the library allows, needs or does not support password.

  When no mode information is passed for METHOD keyword (f.e. NUKE)
  or with no METHOD keyword, all modes are scanned. For every type
  only the highest mode number is used. In case you specify ALL
  keyword, all modes from 0 to 100 are used.

DESCRIPTION
  With xBench you can create benchmarks for xpk sub-libraries. The
  benchmarks created will be relative to the system (hardware and
  possibly software) it is run on and to the file used as data. To
  generate standard xpk-benchmarks shown in information data of the
  sub libraries you will have to run xBench on a A3000/25 with SCRAM,
  using the AmigaVision executable as data.

  xBench uses timer.device for as accurate measurements as possible,
  but the given time results can differ from call to call.

  NOTE: Benchmarks are different for different file-types, so you
  may produce benchmarks for text, exe-files, sounds, ...
```

```
NOTES
  Best method to use is : xBench <filename> TEST PASSWORD TestPwd

  If error messages "FileSize false ..." or "Decrunched buffer ..."
  occur, the library producing these errors should be deleted and
  the library author should be informed of the error.

  When the program crashes, this is most time a problem of a
  sub-library. You can find the bad library when lokking into
  LIBS:compressors. The bad library is most time the one after the
  library that produced last output (when files are sorted
  alphabetically).
  Reboot your system and retest the library with xBench’s METHOD
  option. If it crashes again, delete it!

WARNING
  For accurate measurements xBench have to Forbid() task rescheduling
  while packing and unpacking. This means that multitasking will be
  disabled while xBench is running. Note that doing a Forbid() for a
  long time is potentially dangerous.

THE OUTPUT

  The output data shows as following:

<filename>
Type  Num Version Password  CSize  CTime    CSpd  USize  UTime    USpd Rate
BLZW: 100  3.1              19312   1.37   18280  25044   0.83   30173 22.9

  Type:   the library type
  Num:    the crunch mode which was used
  Version:  version of the library
  Password: when password was used it is shown here
  CSize:    file size after crunching
  CTime:    time used to compress the file (seconds)
  CSpd:   compression speed (bytes/sec)
  USize:    file size before crunching
  UTime:    time used to uncompress the file (seconds)
  USpd:   uncompression speed (bytes/sec)
  Rate:   crunch factor

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR

                Dirk Stöcker
```

## 1.29 xdir

```
                SYNOPSIS
  xDir
  xDir [filenames]
  xDir [directories]
```

```
DESCRIPTION
  xDir shows a listing of all files in the current directory (first
  form), or of a number of files or directories. Wild cards are not
  recognized. Files are sorted alphabetically.

  xDir sums up the uncompressed and compressed total sizes of all
  files shown.

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR

                Dirk Stöcker
```

## 1.30 xloadseg

```
                OVERVIEW
  XPKLoadSeg wedges into LoadSeg() and NewLoadSeg (if available) and
  allows to directly run programs that were compressed using the XPK
  standard. They are decompressed while being loaded. XPKLoadSeg uses
  less than 700 bytes when installed. Should not really bother you.

HOW TO USE
  Just start 'xLoadSeg' from your startup-sequence. No need to 'run'
  it. If you want to remove it, use 'xLoadSeg -q'. The 700 bytes will
  be lost. Do not try to start from Workbench.

HISTORY

  1.0 First release, based on PPLoadSeg by Nico François
  1.1 Complete rewrite: Now able to remove patch
    Symbol and Debug Hunks are now skipped, Overlayed Files
    gracefully fail instead of crashing the machine.

COPYRIGHT
  This program is Public Domain. You may freely use it, spread it,
  enhance it or even delete it. No warranties either expressed or
  implied, use it at your own risk!

AUTHOR

                Christian Schneider
```

## 1.31 xpack

```
                SYNOPSIS
  xPack FILE/M/A,METHOD/K,MINSIZE/N/K,SUFFIX/K,PASSWORD/K,
        ALL/S,FORCE/S,PROGRAM/S,XSCAN/S,LOSSY/S,QUIET/S

DESCRIPTION
  xPack is a command line interface to the XPK compression library.
  It was made to enable you to pack (or unpack) many files quickly
  and comfortably, exspecially for use with the XFH-Handler.
  xPack needs OS 2.04 or newer.

  Main features:
  - supports patterns
  - can scan complete directory trees
  - protection flags, filenote and date of the files are NOT changed
  - packed files will not be repacked by default

  For more details about XPK read the documentation supplied.

ARGUMENTS

FILE    You can supply as many files, directories or patterns
    as you want.
METHOD    the compression algorithm to be used
MINSIZE   All files which are smaller than this value (in bytes)
    will not be crunched (default 512 bytes).
SUFFIX    add/remove supplied suffix if packing/unpacking
PASSWORD  optional Password for encryption (or decryption)
ALL   scan through directory trees
FORCE   Files will be packed even if they are already XPK
    packed and/or their size increases.
PROGRAM   pack only executables (e.g. for
                xLoadSeg
                )
XSCAN   create filenotes for fast directory access with XFH
    (like
                xScan
                )
LOSSY   permit lossy packing
QUIET   No progress report is printed while packing.

  Examples:

   xPack SYS:MetaFont METHOD
                NUKE
                 ALL

  or

   xPack Docs/#?.doc METHOD
                IMPL
                .40 SUFFIX .xpk MINSIZE 1024

  or

   xPack Secret.txt METHOD ENCO PASSWORD Joshua

  or (Decrunch)

   xPack Archive/#?.xpk Archive/#?.pp QUIET

HISTORY
  1.0 (XPKSmart) first internal Release
  1.0 (xSmart)
```

```
    program renamed on a request by
                Urban 'XPK' Müller
                    progress display fits better in the CLI window now
    check for increase of size by packing with XPK implemented
  1.0 program renamed again on a request by
                Urban 'XPK' Müller
                    "FORCE", "PASSWORD" and "SUFFIX" argument implemented
    file handling changed, slower but more secure
    removed Enforcer hit
  1.1 no problems with WShell anymore
    if xPack is started with OS 1.3 a message is printed
     instead of displaying a recoverable Alert
  1.2 added "PROGRAM" parameter
    suffixes may be removed while unpacking
  1.3 wasted my time with a special function for ILBM-files
    added "QUIET" parameter
    "FORCE" is on automatically if a password is supplied.
  1.4 changed "FILE/M" to "FILE/M/A" in template
    added "XSCAN" and "LOSSY" parameter
  1.5 "LOSSY" always active in 1.4

COPYRIGHT
  xPack is free to be spread on public-domain and shareware disks as
  long as they are sold for a reasonable charge that is less than $6.
  This applies not to Fred Fish, he and ONLY he can take more money.
  For use in commercial products the permission of the author is
  required.

AUTHOR

                Matthias Scheler
```

## 1.32 xpk

```
                SYNOPSIS
  xPK [-frsux] [-p password] [-m method] files

       -m = packing method
       -f = force repack
       -s = do not remove original
       -r = recursively (un)pack
       -u = unpack (extract)
       -p = encrypt/decrypt
       -x = pack executables only

DESCRIPTION
  xPK is a command line interface to the XPK compression library.
  It compresses a file using the method given by -m. After the
  process is complete, the original file is removed and replaced
  by its compressed version under the same name.

  The xPK executable can be renamed to a packer name which will
  then be considered as given by -m.

OPTIONS
```

```
  -m = method. After -m you can indicate the name of the packer
       to use, plus a mode number if the packer supports that.
  -f = force. Will enforce packing of already XPK-packed files.
  -s = suffix. Add a .XPK suffix to the compressed version and
       do not remove the original.
  -r = recur. If any directories are encountered, they are packed
       recursively.
  -u = unpack. Will unpack the indicated files. (same as -e).
  -p = password. Will be used for encryption or decryption.
  -x = executables. Will refuse to pack files that are not
       executable or are overlaid. For use with xLoadSeg.

EXAMPLES
  xPK -rm NUKE dh1:modules
  xPK -m  IMPL.50 df0:OVERVIEW
  xPK -xm NUKE dh4:
  xPK -r -p topsecret -m FEAL.32 dh1:private_docs

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR

                Urban Dominik Müller
```

## 1.33 xquery

```
                SYNOPSIS
  xQuery
  xQuery [packer]

DESCRIPTION
  xQuery shows important parameters about a packer, or if
  none indicated, all packers.

EXAMPLE
  xQuery FEAL

  Packer : FEAL
  Name   : Fast Encryption ALgorithm 1.0
  Descr. : FEAL-N with CBC1.  Password protects data with selectable safety.
  DefMode: 16
  Mode   :   0..4     5..8     9..16   17..32   33..100
  Descr. :  fastest   fast     safe     safer   safest
  PkSpeed:  238 K/s  171 K/s  109 K/s   63 K/s   34 K/s
  UpSpeed:  244 K/s  174 K/s  109 K/s   63 K/s   34 K/s
  Ratio  :    0 %      0 %      0 %      0 %  0 %

  The meaning of the fields:

  Packer : The 4-letter name of the packer
  Name   : The full packer name
  Descr. : The packer description
  DefMode: The default mode
  Mode   : Below information is valid for this range of modes
  Descr. : Mode description
  PkSpeed: Packing speed for this mode range
  UpSpeed: Unpacking speed for this mode range
  Ratio  : Compression factor (higher=better)

  All timings were measured on an A3000. Divide by 5 to get
  timings for 68000.

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR

                Urban Dominik Müller
```

## 1.34 xtype

```
                SYNOPSIS
  xType filenames

DESCRIPTION
  Prints the given files to stdout, decompressing them if they are
  compressed.

EXAMPLE
  xType intuition.doc.nuke

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR

                Urban Dominik Müller
```

## 1.35 xup

```
                SYNOPSIS
  xUp [-s] [-S] [-p password] filenames

DESCRIPTION
  xUp unpacks the given files, replacing the original by the
  uncompressed version. Wild cards are not supported, and file
  attributes are not yet preserved.

OPTIONS
  -s = suffix. Keeps compressed version (when suffix is added),
       stores uncompressed version under the same name minus .xpk
       suffix.
  -S = Same as -s, but removes any .#? suffix and not only .xpk one.
  -p = password. Uses given password for decompression

EXAMPLE
```

```
  xUp -p secret mytext

COPYRIGHT
  Freely distributable for noncommercial use.

AUTHOR

                Dirk Stöcker
```

## 1.36 xscan

```
                SYNOPSIS
  xScan FILE/M/A,ALL/S,REMOVE/S

DESCRIPTION
  xScan is a small CLI command which scans through XFH partition and
  modifies them. After those modifications XFH (V1.34 or newer) will
  be able to read directories MUCH faster. In fact you will no more
  notice a speed difference between XFH and the normal FileSystem.

  VERY IMPORTANT:
  xScan will NOT work if you use it directly on a XFH partition, it
  will just do nothing. Instead of that you must use it on the
  PHYSICAL directory of the XFH partition. E.g. if your XFH partition
  is called "XH0:" and the rootdir of is "DH0:Archive", DO NOT use
  "xScan XH0: ALL" but "xScan DH0:Archive ALL".

THEORY
  How does "xScan" work ?

  If XFH scans through a directory it opens EVERY file to check if
  it is packed or not. That's why it is so slow.
  xScan scan once through the directories for files. If it finds one
  with an UNUSED filenote it adds a special one (filenote) to the file.
  This filenote contains an ID string, some check values and the length
  of the unpacked file(*).
  If the new XFH scans through the directories it checks for such
  filenotes and after finding one with still valid check values it will
  take the unpacked length from the filenote without opening the file.
  That's why the new version is faster. Of course these special
  filenotes will be hidden.

  (*) I do not want to explain the format exactly, because people
      should not use these informations.

ARGUMENTS
  FILE:   You can supply as many files, directories or patterns
      as you want.

  ALL:    scan through directory trees

  REMOVE:   remove filenotes instead of creating them

  Examples:
   xScan SYS:Archive/MetaFont ALL

  or

   xScan Docs/#?.doc REMOVE

HISTORY
  1.0 initial release for XFH 1.34
  1.1 adds special filenotes to unpacked files, too
  1.2 does NOT follow softlinks any more

COPYRIGHT
  xScan is free to be spread on public-domain and shareware disks as
  long as they are sold for a reasonable charge that is less than $6.
  This applies not to Fred Fish, he and ONLY he can take more money.
  For use in commercial products the permission of the author is
  required.

AUTHOR

                Matthias Scheler
```

## 1.37 contacts

```
                Please remember, that some of these addresses may be false, so do ↩
                   not
blame, if you do not get answer. If you get newer information, please
contact me (the first one).

Autors of the main xpkmaster system (and some additional stuff).
Contact in the given order!


                Dirk Stöcker

                Christian von Roques

                Urban Dominik Müller

                Bryan Ford
                Autors of Sublibraries:

  André Beck
                IDEA

                Karsten Dageförde

                RAKE

                Stephan Fuhrmann

                DLTA

                Martin Hauner
                HFMN

                John Hendrikx

                SQSH

                Zdenek Kabelac

                MASH

                Jorma Oksanen

                SMPL
                , FRLE,
                HUFF

                Christian von Roques

                FAST
                ,
                FEAL

                Peter Struijk

                IMPL

                Marc Zimmermann

                HUFF
                Translators:
  Deutsch
                Dirk Stöcker
                  Español Dámaso D. Estévez <amidde@arrakis.es>
  Français  Georges 'Melkor' Goncalves <melkor@lords.com>
    Laurent Kempé <lkempe@nucleus.fr>
  Italiano  Dario Manzoni <dmanzoni@spin.it>
  Nederlands  Frits Letteboer <dagraver@dds.nl>
  Norsk   Kim Roar Utsi <kimme@arcticnet.no>
  Polski  Marcin Orlowski <carlos@inet.com.pl>
  Russian Oleg Sergeev <bigblack@neworder.spb.ru>
  Srpski  Ljubomir Jankovi <lurch@afrodita.rcub.bg.ac.yu>
  Svenska Andreas Pålsson <did@algonet.se> [version 3.11]
  ÃeÓtina VÉt óindlÁÒ <xsindl00@stud.fee.vutbr.cz>

Other related persons:

  Harmut Goebel   Oberon Interface & examples

                Matthias Meixner
                  xpkarchive.library,
                SHRI

                Kristian Nielsen
                  XFH

                Nicola Salmoria
                  XFH commodity
                Matthias Scheler
                  XFH,
                xPack

                Christian Schneider
                  XPK concept,
                xLoadSeg

                Jan Schwenke
                    HotHelp files
  Markus Wild   GCC interface & examples

And surely there are a lot of persons I forgot.
```

## 1.38 contact dirk stöcker

```
Name:   Dirk Stöcker
Address:  Geschwister-Scholl-Straße 10
    01877 Bischofswerda
    GERMANY
Telephone:  GERMANY (+49) (0)3594 706666
E-Mail:   stoecker@rcs.urz.tu-dresden.de
    stoecker@amigaworld.com
WWW:    http://home.pages.de/~Gremlin/
    http://www.amigaworld.com/support/xpkmaster/
```

## 1.39 contact christian von roques

```
Name:   Christian von Roques
Address:  Forststrasse 71
    76131 Karlsruhe
    GERMANY
Telephone:  GERMANY (+49) (0)721 621253
or
Address:  Kastanienweg 4
    78713 Schramberg
    GERMANY
Telephone:  GERMANY (+49) (0)7422 53822
E-Mail:   roques@pond.sub.org
    roques@ipd.info.uni-karlsruhe.de
    roques@ira.uka.de
```

## 1.40 contact bryan ford

```
Name:   Bryan Ford
Address:  8749 Alta Hills Circle
    Sandy, UT 84093
Telephone:  (801) 585-4619
E-Mail:   bryan.ford@m.cc.utah.edu
```

```
    baf0863@cc.utah.edu
    baf0863@utahcca.bitnet
```

## 1.41 contact urban dominik müller

```
Name:   Urban Dominik Müller
Address:  Schulhausstrasse 83
    CH-6312 Steinhausen
    SWITZERLAND
E-Mail:   umueller@indiac.relog.ch
    umueller@amiga.icu.net.ch
    umueller@amiga.physik.unizh.ch
    umueller@iiic.ethz.ch
```

## 1.42 contact karsten dageförde

```
Name    Karsten Dageförde
E-Mail:   dagefoer@rzcipa03.rz.tu-bs.de
    dagefoer@ibr.cs.tu-bs.de
    dagefoer@rob.cs.tu-bs.de
    K.Dagefoerde@tu-bs.de
```

## 1.43 contact stephan fuhrmann

```
Name:   Stephan Fuhrmann
Address:  Ostmarkstraße 19
    76227 Karlsruhe
    GERMANY
E-Mail:   Stephan.Fuhrmann@stud.uni-karlsruhe.de
```

## 1.44 contact martin hauner

```
Name:   Martin Hauner
Address:  Max-Born-Straße 5
    38116 Braunschweig
    GERMANY
E-Mail:   drizzt@trashcan.escape.de
```

## 1.45 contact john hendrikx

```
Name:   John Hendrikx
Address:  Figarostraat 36
    3208 PD   Spijkenisse
    The Netherlands
E-Mail:   FIDO: 2:285/813.8
```

```
    AMY:  39:153/201.8
    NLA:  14:101/200.8
```

## 1.46 contact zdenek kabelac

```
Name:   Zdenek Kabelac
Address:  Policna 135
    75701 Valasske Mezirici
    Czech Republic
E-Mail:   kabi@informatics.muni.cz
WWW:    http://www.muni.cz/~kabi/
```

## 1.47 contact jorma oksanen

```
Name:   Jorma Oksanen
Address:  Hämeentie 6-8 A 4
    13200 HÄMEENLINNA
    FINLAND
E-Mail:   tenu@sci.fi
Telephone:  FINLAND (+358) (3) 6120 217
```

## 1.48 contact jan schwenke

```
Name:   Jan Schwenke
Address:  Dorfstraße 55
    09465 Cranzahl
    GERMANY
E-Mail:   jsc@fh-zwickau.de
```

## 1.49 contact peter struijk

```
Name:   Peter Struijk
Address:  Veulenkamp 28
    2623 XD  DELFT
    The Netherlands
E-Mail:   winfjmf@dutiws.twi.tudelft.nl
```

## 1.50 contact marc zimmermann

```
Name:   Marc Zimmermann
E-Mail:   zimmerma@ibr.cs.tu-bs.de
```

## 1.51 contact matthias meixner

```
Name:   Matthias Meixner
Address:  Sandberg 13
    36145 Schwarzbach
    GERMANY
E-Mail:   meixner@rbg.informatik.th-darmstadt.de
```

## 1.52 contact kristian nielsen

```
Name:   Kristian Nielsen
Address:  Groenjordskollegiet
    room 6111
    Groenjordsvej
    DK-2300 Koebenhavn S
    Denmark
E-Mail:   bombadil@diku.dk
```

## 1.53 contact nicola salmoria

```
Name:   Nicola Salmoria
Address:  Via Piemonte 11
    53100 Siena
    ITALY
E-Mail:   MC6489@mclink.it
```

## 1.54 contact matthias scheler

```
Name:   Matthias Scheler
Address:  Schützenstraße 18
    33178 Borchen
    GERMANY
Telephone:  GERMANY (+49) (0)5251 399031
E-Mail:   tron@lyssa.pb.owl.de
    FidoNet: Matthias Scheler 2:243/6310.10
```

## 1.55 contact christian schneider

```
Name:   Christian Schneider
Address:  Im Schilf 15
    CH-8044 Zurich
    Switzerland
E-Mail:   BIX:    hschneider
    Internet: cschneid@amiga.physik.unizh.ch
```